

Game Physics FINAL

by Jacob James-MacHin

Submission date: 05-Dec-2024 08:53PM (UTC+0000)

Submission ID: 244829930

File name: 50542_Jacob_James-MacHin_Game_Physics_FINAL_628791_864745531.docx (999.35K)

Word count: 2497

Character count: 12887

GAME PHYSICS AND HOW THEY WORK

Games Architecture and Mechanics Assignment 1

By Jacob
James-Machin

Contents

Introduction	2
The Fundamentals of Games Development	2
Numerical Integration Methods used in gaming.....	2
Explicit Euler.....	3
Advantages.....	3
Disadvantages	3
Velocity Verlet.....	3
Advantages.....	4
Disadvantages	4
Physics simulation	4
Games Code.....	5
Euler's Method	6
Velocity Verlet.....	9
Conclusion	10
References.....	10

Introduction

Physics in Video Games are commonplace. Going from 2D to 3D Games. Usually, they are emulating real life physics. And some physics engines are integral to the game and the design of the game.

Every game we can think of uses physics in one form or another. Its usually built into the design and simulation of said game. Every game uses many various forms of physics. Going through object based or ragdoll physics and it helps add to the gameplay of said game. It is also built into the engine of said game. For instance, every game with a 3D engine would have physics implemented.

One of these engines is the Source Engine [1]. Created in 2004 the source engine's main selling point is the real-life simulation of physics.

And it works. The games that run on the Source Engine like Half-Life 2 or Portal have puzzles that involves physics and our understanding of physics. [2], [3] The games make it out to be rather simple like we can pick up objects and toss them around and they would react in a way we would expect them to. However, the Maths involved is rather complicated. And to create these games and the engine it runs on we need to understand how physics works on not just an aesthetic value but in a mathematical value.

So, it is important to consider the maths and methods behind simulating physics in a game engine. Since if we want to design a game centered around physics then the maths behind physics then understanding the methods is key.

The Fundamentals of Games Development

The Fundamentals of Games Development is having a bare minimum grasp on maths and numerical integration methods. Understanding these as a developer is part of the process of game making. As mentioned above numerical integration is vital to simulate game physics in a realistic manner. How can we simulate the physics without knowing the maths behind the physics.

It is the kind of maths that would look scary on paper (which it does) but once we are able got the grasp, we would be able to code it in and make it work naturally. And its not too hard to understand. For example, anyone who is bad at maths can know what X, Y, Z would mean. Despite being the basic part once we got that means that we are on the right path.

Numerical Integration Methods used in gaming.

Numerical Integration is a technique we as developers use to that approaches mathematical problems that involves integration and computation of values within programming. Like if we want to create realistic detailed physics simulations portrayed in games like in Half-Life 2, we need to utilize the numerical integrations that are used behind it. These include things like Gravity, mass, velocity etc.

When it comes to programming or simulating video game physics there are numerical integration methods to consider. There are several Numerical Integration methods used in programming. They are seen through three orders. First Order Integrators, Second Order Integrators and Higher Order Integrators

When we are trying to implement a physics engine within a game it involves a ton of high-level maths. However, to get started we can look at Newton's equations of motion such as $\text{Force} = \text{Mass} * \text{Acceleration}$ or $\text{Acceleration} = \text{Force} / \text{Mass}$

Here are the numerical integration methods that developers use in Games Development [4]

To solve problems like this we need to approach these issues breaking them down into simple and easy to understand equations. For instance, if we want to understand how physics work.

Below are the most used examples of numerical equations used in programming.

Explicit Euler

This method is most used by Developers. It involves using an initial position and velocity on a particular object for example a race car and it calculates the next position and velocity as the race car goes long the track. This is calculated through a time step value, and it would be represented by "dt."

Here is the calculation....

$$\text{Velocity}_{n+1} = \text{Velocity}_n + (\text{Acceleration} * dt)$$

And to find the Velocity we use this method.

$$\text{Position}_{n+1} = \text{Position}_n + (\text{Velocity}_n * dt)$$

However, this method does have its Advantages and Disadvantages. These Include:

Advantages

This method is the most simple and easy to use. So, if we are starting out and making a game using this method would be ideal as it lets us grasp the concept without delving too much into the details of how physics work.

Disadvantages

However, due to its simplicity its rather inaccurate which can be an issue when we are trying to replicate real life physics. Going back to my race car game example. These racing games used a ton of accurate and precise physics simulation and using and Explicit Euler may miss the precise details required.

Velocity Verlet

Velocity Verlet is another algorithm used to calculate the velocity using Newton's Formula. To use this formula, Calculate the position, time, and acceleration. This is considered the central equation for Newton's formula. We can use this as part of force integrators like Verlet Integrators. [5]

Here are the equations.

$$\text{Velocity}_{n+1/2} = \text{Velocity}_n + \text{Acceleration}_n * dt/2$$

$$\text{Position}_{n+1} = \text{Position}_n + \text{Velocity}_{n+1/2} * dt$$

$$\text{Acceleration}_{n+1} = \text{Force}/\text{Mass}$$

$$\text{Velocity}_{n+1} = \text{Velocity}_{n+1/2} + \text{Acceleration}_{n+1} * dt/2$$

Advantages

The biggest advantage for this method is that it can simulate with more accuracy, and we can utilize the two verlet techniques which is essentially a more kinetic equation to a motion of an object.[6]

Disadvantages

However, the biggest disadvantage is that compared to Explicit Euler its much less efficient despite being more accurate. It requires more equations that accounts for the additional questions.

Physics simulation

On this next part we will need to simulate a race car using a mathematical equation that calculates the speed/velocity and the gravitational force of the car. And for this we will use Explicit Euler as it is the simplest numerical integration as a lot of game developers use in their code

To do this it is necessary to use **$\text{Position}_{n+1} = \text{Position} + (\text{Velocity} * dt)$** . Which would help represent the simulation of a car moving.

To understand this equation, we need to approach these several mathematical problems.

position (y). (This is where the car is located on the road)

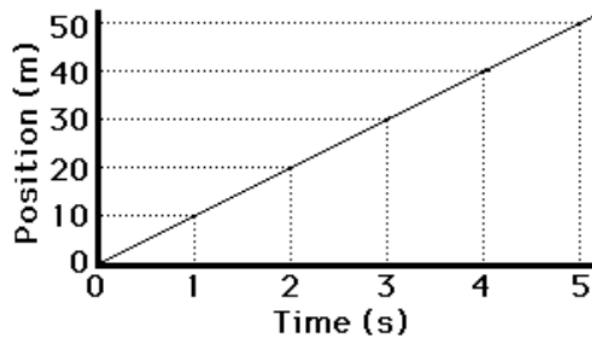
Velocity (dy/dt) which determines how fast we are going and how the position is changes over time hence dt means speed and y is position.

Acceleration (d^2y/dt^2) determines how fast our speed and how that speed changes as we are accelerating.

Time(t) as mentioned above this can be represented dy/dt when we calculate how fast a car is moving. The time (t) determines how long it would take for the car to reach its destination.

This plays a part with how we would simulate driving a race car since we need to consider position, speed, and acceleration.

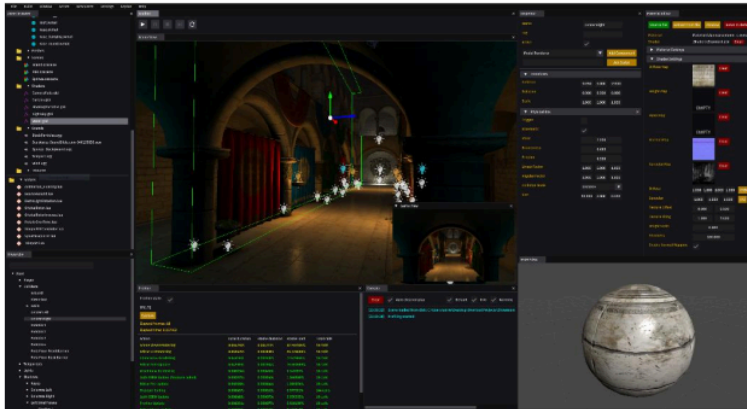
There are a ton to consider when calculating various aspects of driving a car, but Velocity, Position, time, and Acceleration is the basics of understanding car simulation.



[7]



Understanding this concept will let us understand how developers would implement physics in other areas such as collision or fluid dynamics or gravity physics. And the best thing is that they do not require an understanding of complex maths. But rather how we would implement this in C++. Many developers say that they did not use much maths to create their games [8] [9]



However, when it comes to Game Engines that is an entirely different story as Game Engines do require complex maths and understanding complex equations to build a game engine from the ground up. Especially for a game engine that requires physics and realistic physics simulation. Game Engines like the Source Engine, Unity, Unreal and etc utilize physics which uses numerical integrations. Its part of Games Development.

Games Code

To simulate this in code is rather simple. As programming requires using a set language that the computer would understand so we would only need to use the programming language such as C++ to represent and simulate this equation. The reason using C++ is the same reason any other developer would use it. Performance and efficiency. C++ provides good control over the system's resources like memory or CPU usage and that is important if we want to program a game considering a game would utilize those aspects [10]

Here is what is possible to do in C++ to simulate car physics using two numerical integration methods which is possible to have it displayed on a graph.

Euler's Method

To do the Euler's method, we need to start out linking the code that refers to a file writer which can allow us to file write and print out the graph using the "include" function.

```
#pragma once
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "CarPhysics.h"
```

By naming it "CarPhysics.h" and in that header file there is code that we need to write that would print these functions out in a excel sheet. Also using "string," "fstream," "iostream," "iomanip" which allows me to use more functions like strings or allow us to use maths using floats. (Float allowing me to make a number that changes)

```
//This code is supposed to represent car physics
int main()
{
    //the code below establishes the variables needed for the car to move

    float position = 0; //This establishes position as a float

    float time = 0.f; //this is establishes time as a float and as the float function should change. As time is supposed to change
    float deltaTime = 0.01; //deltatime like float time its supposed to change the time by 1 second
    float distance = 0; //this would be the distance. It would change as the car moves
    float velocity = 0; //the velocity would change with the speed of the car
    float acceleration = 9.81; //acceleration would increase as the car goes faster

    const float coefficientRR = 0.21; //this is the coefficient rolling resistance
    const float gravity = 9.81; //This is the gravity which would remain the same
    const float mass = 1; //this is the mass of the vehicle which
    const float AirDensity = 1.2; //the air density would change as the car increases speed
    const float dragCoefficient = 0.34; //drag is supposed to represent the drag of the car as it increases in speed. This is related to coefficient
    const float frontal = 0.01; //the frontal is the front of the car
```

The next part of the code as displayed in this screenshot here will establish the various float functions that is needed for the Euler's equation that would help calculate the spring car's movement.

To do this, Variables should be made such as Position (as the car moves the position changes), Time (the time would change when the car moves), "deltaTime" (time moving by 1 second), distance (would change as the car position changes), velocity (which would change depending on the speed) and Acceleration (which would increase as the car goes faster)

"Const Float" however is a float that would not change but would be taken in account during this program. With things like rolling and drag coefficient which the rolling resistance and drag resistance of the car. Air Density which would change as the car increases speed. Using the constant value. Mass would be the overall weight and mass of the car and the Frontal which would be the frontal speed of the car.

```

filewriter fw("CarPhysics results to print.csv");

//this writes the results into a excel sheet so I can display it in a graph

```

As mentioned above I have written code that will print the spring car results in an excel sheet where it would be printed out in a graph.

```

while (velocity > 0 || time == 0) //this is where the code happens within the while loop. It will loop when I determine when the function ends
{

    float force = 1.2 * (16 - position); //this is float force. Basically sets the force as 1.2 and its subject to change
    if (position > 6)
    {
        force = 0; //If distance travelled, x is greater than 6 metres then set Ft to zero Newtons (see equation in the document).
    }

    float fD = coefficientRR * mass * gravity + (dragCoefficient / 2) * AirDensity * frontal * (velocity * velocity); //this is the main calculation

    acceleration = (force - fD) / mass; //This code would calculate the force and the calculation written above
    velocity += acceleration * deltaTime; //this determines that if the velocity and acceleration changes the time also changes as well
    position += velocity * deltaTime; //same here but for the position as well as its tied to velocity

    //This is Euler's method
}

```

This here is the main bulk of the code. Where the Euler's Method equation is played out within the code. We need to place this function into a "While Loop" so it can determine where the code will stop rather than stopping as is.

To do this, like in the screenshot establish the float force as 1.2 and times it by 16 then – position with Force = 0. Doing this means that if the distance travelled is greater than six meters then set the force to zero as its required by the Euler's Method equation.

"Float fD = coefficient * mass * gravity + (dragCoefficient / 2) * AirDensity * frontal * (velocity * velocity);" is basically the main equation aka **" $F_d = CR \cdot mg + \frac{1}{2} CD \cdot AV^2$ "** simplified using c++

Finally, the rest of the code would calculate the force then determine that if the velocity and acceleration changes while the time also changes as well. Along with the position as well as its velocity

```

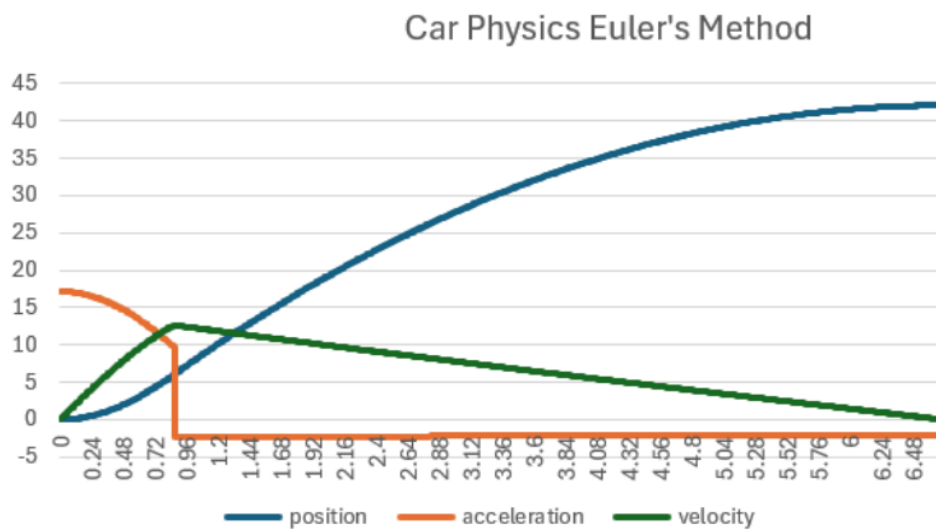
64
65     fw.write(time, velocity, acceleration, position);
66
67     //this code is supposed to be the main algorithm for how the time would change as the car moves
68
69
70
71
72     time += deltaTime; //This is a fix that helps change the time.
73
74
75 }
76
77
78
79
80
81
82
83 return 0; //this ends the function
84
85

```

And finally, here this is code ends the function. The function ends using “fw. write” code to display the graph.

(To note time += deltaTime is basically a fix that implemented that would allow the time to change through 0.01 second intervals)

And here is the finalised graph.



Velocity Verlet

To start the Velocity Verlet code here is what is required to do so.

```
case 2:
//This is Velocity Verlet

float force = 1.2 * (16 - position); //This is a float that establishes the force times the position
float fD = coefficientRR * mass * gravity + (dragCoefficient / 2) * AirDensity * frontal * (velocity * velocity); //This means the fD is part of the main calculation
float accelerationNew(); //I'm establishing a new variable for the velocity verlet function
```

To accomplish this, we need to reuse some code and establish new code. As shown here with "AccelerationNew" This is a new variable for the Velocity Verlet function, and it will calculate a new acceleration as per the requirement of the calculation. Acceleration does play a significant role in this algorithm.

The next step is to place in the While Loop.

```
while (velocity > 0 || time == 0) //this is where the code happens within the while loop. It will loop when I determine when the function ends
{
    velocity = velocity + acceleration * (deltaTime / 2); //velocity being part of acceleration and deltaTime

    acceleration = (force - fD) / mass; //acceleration equalling the force and fD divided by mass
    position = position + velocity * deltaTime; //position plus velocity times deltaTime
    if (position > 6)
    {
        force = 0; //If distance travelled, x is greater than 6 metres then set Ft to zero Newtons (see equation in the document).
    }
}
```

This is reused partially from the Euler's Method function but ideally this would set this function in a continuous loop that would end when a certain standard has met. For example, when a certain position/velocity threshold has been reached the function would end then it can print it on a excel graph. Furthermore, it is possible to adjust the code to adjust for the Verlet method.

```
//below is the main calculation for Velocity Verlet
velocity = velocity + acceleration * (deltaTime / 2); //Velocity plus the acceleration times deltaTime divided by 2
position += velocity * deltaTime + 0.5 * acceleration * deltaTime * deltaTime; //Also part of the main Velocity Verlet function
velocity += 0.5 * (acceleration + accelerationNew) * deltaTime; //velocity adding value to 0.5 with acceleration plus accelerationNew times Delta Time
acceleration = accelerationNew; //Acceleration new being part of Velocity Verlet which would be an updated function to the acceleration

fw.write(time, velocity, acceleration, position); //this prints the code into a graph

time += deltaTime; //this updates the time
```

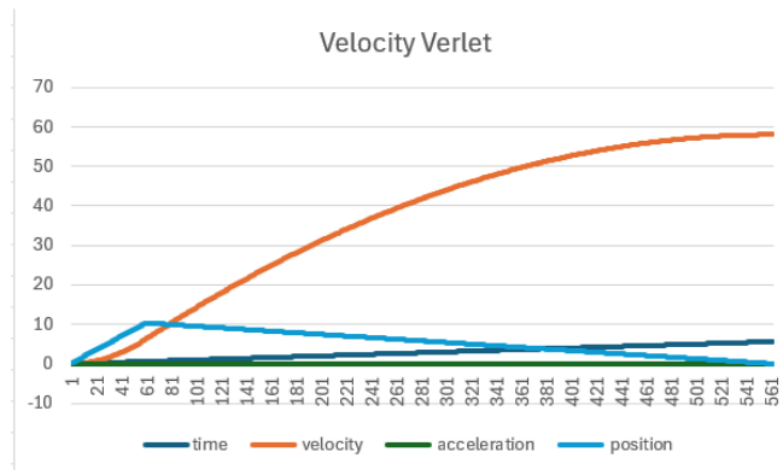
Next would be the calculation for the Velocity Verlet. To start this, we need to add Velocity with the acceleration time then divide it by two. This means that when the function is calculating the spring car's movement it would divide the time it is taken for the velocity to slow down.

Next is the position. The position will change depending on the velocity with a Delta Time interval of 0.5. This would allow the position, velocity, and Delta Time to change at the same time.

Having the velocity to change with the interval of 0.5 it can allow for the new acceleration and delta time to be part of the equation.

The new acceleration along with acceleration would allow for both the velocity and position to be adjusted accounting for the velocity verlet function.

Finally, the time and "deltaTime" allows the time to change during this equation as demonstrated in in the Euler's Method.



Then finally the fw. write allows the code to be printed into a graph using the same system that I had for Euler's.

Conclusion

To conclude this paper. The better understanding of game physics using numerical methods is a core part of Game's Development and the understanding of physics. Every game around has an element of physics meaning that these numerical functions are a part of any game we play. From advanced physics simulations in the Source Engine to simple spring car simulations. Physics is everywhere in gaming. Furthermore, using C++ to simulate physics aids in the development of games especially when you integrate it into the game engine. Its why most games especially big physics heavy games are written in C++.

References

- [1] "Source (game engine)," *Wikipedia*, Mar. 18, 2020. [https://en.wikipedia.org/wiki/Source_\(game_engine\)](https://en.wikipedia.org/wiki/Source_(game_engine))
- [2] "Half-Life 2," *Wikipedia*, May 25, 2020. https://en.wikipedia.org/wiki/Half-Life_2
- [3] Wikipedia Contributors, "Portal (video game)," *Wikipedia*, Jul. 05, 2019. [https://en.wikipedia.org/wiki/Portal_\(video_game\)](https://en.wikipedia.org/wiki/Portal_(video_game))
- [4] jdickinson91, "Numerical Integration in Games Development," *Understanding Games Development*, Jan. 22, 2015. <https://jdickinsongames.wordpress.com/2015/01/22/numerical-integration-in-games-development-2/>

[5] J. Schloss, "Verlet Integration · Arcane Algorithm Archive," www.algorithm-archive.org.
https://www.algorithm-archive.org/contents/verlet_integration/verlet_integration.html

[6] Idézet egy cikkből: The Verlet algorithm." Available:
<https://www.hds.bme.hu/letoltesek/targyak/BMEGEVG1MOD/verlet.pdf>

[7] "Position-Time Graphs: Meaning of Shape," www.physicsclassroom.com.
<https://www.physicsclassroom.com/class/1DKin/Lesson-3/The-Meaning-of-Shape-for-a-p-t-Graph>

[8] "How is C++ Used in Game Development: The Secret to Stellar Gaming Experiences," *Teamcubate.com*, 2024. <https://teamcubate.com/blogs/how-is-c-plus-plus-used-in-game-development> (accessed Nov. 21, 2024).

[9] "Fundamental Math for Game Developers," *pikuma.com*. <https://pikuma.com/blog/math-for-game-developers>

[10] "Why is Math Important to Make a Career as a Game Designer," www.jetlearn.com.
<https://www.jetlearn.com/blog/math-for-game-design>

Game Physics FINAL

GRADEMARK REPORT

FINAL GRADE

56/100

GENERAL COMMENTS

Engagement with Literature: 7/15

Knowledge and Understanding: 7/15

Cognitive and Intellectual Skills: 10/20

Practical Skills: 30/45

Transferable Skills: 2/5

Overall 56/100

Comment:

See inline comments also

PAGE 1

PAGE 2

PAGE 3



Comment 1

In academic writing, particularly in scientific and technical papers, the use of the first person (e.g., "I", "we") is generally discouraged as it can make the writing appear subjective and less formal. The focus of academic work should be on the research, analysis, or findings rather than the researcher. Using an objective tone ensures that the argument or evidence takes precedence over personal opinions or experiences, which enhances the perceived credibility and professionalism of the work. For example, instead of saying, "I conducted the experiment," you could write, "The experiment was conducted," which maintains an impersonal and objective stance. To improve in this area, aim to use passive voice judiciously or rephrase sentences to emphasise the action rather than the actor. Additionally, consider adopting phrases like "the research indicates" or "it was observed" to convey findings without self-referencing. Reviewing published papers in your

field can provide valuable examples of how to maintain an objective tone while ensuring clarity and precision.



Comment 2

It's a bit informal for a scientific report



Comment 3

A strong assumption!



Comment 4

Good, this would have been an ideal point for citation.

Referencing is important because it shows where you got your information from. It gives credit to the original authors for their ideas or research and helps others find the sources if they want to learn more. It also makes your work more trustworthy by showing that you have based your findings on reliable research. Without proper references, readers might not know if your information is accurate or where it comes from.

PAGE 4



Comment 5

Another ideal candidate for a reference, as this is referring to the work of another scientist you should be giving him credit!



Comment 6

DT being delta time would have been good to state.



Comment 7

Labelling an equation as a figure is typically done to maintain consistency with the formatting style used in the document. In some academic styles, such as in certain fields of science and engineering, equations are treated as visual elements or figures, especially when they are central to the content of the paper. By labelling an equation as a figure, it allows for clear referencing throughout the text and ensures that it is easily distinguishable from the main narrative. This approach helps readers locate and focus on key mathematical expressions more effectively.



Comment 8

Another ideal candidate for a reference, who stated this? How can it be proved or disproved?

Where is the evidence?

PAGE 5

PAGE 6



Comment 9

what is this random reference to? if it is to the screenshot you need to use a table of figures for that.

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12
