Creating a physics Library for use in mobile app development.

By Michael Shalaby
Supervised by Keith Mannock
Student ID: mshala02
Msc Computer Science Project Proposal

Abstract:
This proposal will outline the design requirements of to create a Physics Library for mobile game development using open source graphics libraries and rendering libraries. The proposed library will be built using kotlin as the language of choice in order to specialise the physics library for android app development. In order to assess the viability of the physics library, the library will require a lot more than unit testing to see if it works. Therefore the ideal testing ground for this library would be to use it to simulate the daring dam-busters raid "Operation Chastise" carried out by the british during WW2.

Disclaimer:
This proposal is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service.

Contents

History of physical simulations in video games.

Since the introduction of video games in the 1970s, the first physics library came in the form of collision detection as seen in the game Pong (1972). Since then video game physics have seen a lot more development in recent years. These improvements vary by simulating variable motion in earlier racing games as demonstrated in pole position released 1982 to demonstrate how higher speed turning would lead to a larger turning radius to demonstrating the difference in driving on different surfaces like driving on tarmac then driving on grass or gravel.

Over the following decades, more and more physical properties to video games became more and more mainstream. These would range from more simple applications like detecting collision between two separate objects and the transition from using ray tracing to simulate gun fire to implementing projectile motion to simulate the motion of a projectile like a bullet as travels through the air, calculating its new position at every iteration of time.

Why build a new physics library for mobile apps

While there are many excellent physics libraries available with a few being suitable for android app development including jbullet, a physics engine ported to java. There isn't currently an existing physics library built with Kotlin. While java and kotlin both use the same compiler, the main issue java as a programming language face is the fact that it typically uses a lot more boilerplate code than kotlin.

As kotlin has become considered to be the default language for android development since the release of android studio 3.0 has led to popular IDE's like Jetbrain's IntelliJ to utilise a kotlin to java converter to help developers reduce the amount of code they need to write and create less error prone programs for android in comparison to using Java.

While Java is still being updated today with the latest release of Java 12. It is still considered by many to not be a fully modern programming language whereas kotlin is being viewed more and more as the successor to Java for android development.

Why simulate Operation Chastise:

During my undergraduate studies in 2013, one of my projects was to build a web app to simulate some aspects of the bouncing bomb operation. However, due to having an extremely limited understanding in programming, the amount of physical properties that was simulated was very limited. However, as my understanding in programming has improved, so has my understanding on how to implement a more realistic simulation of a bouncing bomb by using less constants like having the velocity of the bomb along the i direction dynamically be affected by skimming the surface rather than a flat value of velocity reduced every time it skims.

A brief history of operation chastise

During the night time on the 16th and 17th of May, the RAF launched one of the most daring bombing raids to destroy three dams located by the industrial heartland of Germany. The reason why dams where chosen in specific was because it was believed that by destroying the dams, it would flood the surrounding areas around the dam and prevent any electricity being generated by the dams. The idea behind this was that by destroying the dams, the British could cripple Germany's industrial capabilities a lot more effectively than launching conventional bombing raids directly over the city.

What makes operation chastice the ideal testing ground for the physics library

As the dam where protected by anti-torpedo nets. The bombs dropped where designed to bounce across the lakes in a similar manner to 'skipping stones'. Thus, negating the effect of torpedo nets underwater. Therefore, this particular event provides many interesting physical aspects to simulate. This would include:

- The height in which the bomb is dropped. During the original operation, the plane was required to fly 60ft above the surface of the water. Therefore, the user can be given the option to adjust the height in which the bomb is dropped from.
- The rate in which the bomb was spinning. During the original operation, the bouncing bomb was required to have a back spin at 700 rpm to increase the speed in which the bottom of the bomb would hit the water. Therefore the user could be given the option to change the rate in which the bomb is spinning in order to see

The physics Library should be able to do:

- Simulate the linear and Rotational Motion of an Object to simulate the motion of the bouncing bomb
- Simulate Collisions between objects and accurately simulate the result of collision in order to simulate the bouncing bomb colliding with the surface of the water and wall of the dam
- Simulate physical properties of both Soft and rigid body objects to simulate how the bouncing bomb will interact with surfaces of different properties i.e. water, concrete or more outlandish surfaces like grass or gravel.
- Be able to incorporate different shapes for different objects with different properties i.e. would the bomb be as effective if it was an oval cylinder rather than a circular cylinder.
- Be able to simulate multiple objects independent of each other in a realistic setting i.e. there was a lot more than one plane involved and therefore a lot more than one bouncing bomb being launched.

Roadmap:

1. Start with deriving and programming linear motion
2. Using the methodology to derive linear motion, apply it to demonstrate how rotational motion is derived
3. Understand how to implement linear motion into usable code through the use of vectors
   i.e. for 2d motion: $\text{magnitude of velocity} = \sqrt{\mathrm{i}^2 + \mathrm{j}^2}$ or for 3d motion $\text{magnitude of velocity} = \sqrt{\mathrm{i}^2 + \mathrm{j}^2 + \mathrm{k}^2}$(if applicable due to limitations in my current programming abilities
4. Implement trig equations for the calculation of angles
5. Understand how to implement rotational motion through the use of matrix algebra to simulate the rotation of an object
6. Implement boundary mechanics for objects, this should incorporate properties like:
   a. Rigid body properties
   b. Soft body properties
7. Implement a collision model between objects of different boundary mechanics
   this should include:
   a. Collision between different object with different physical properties properties
   b. Collisions at an angle

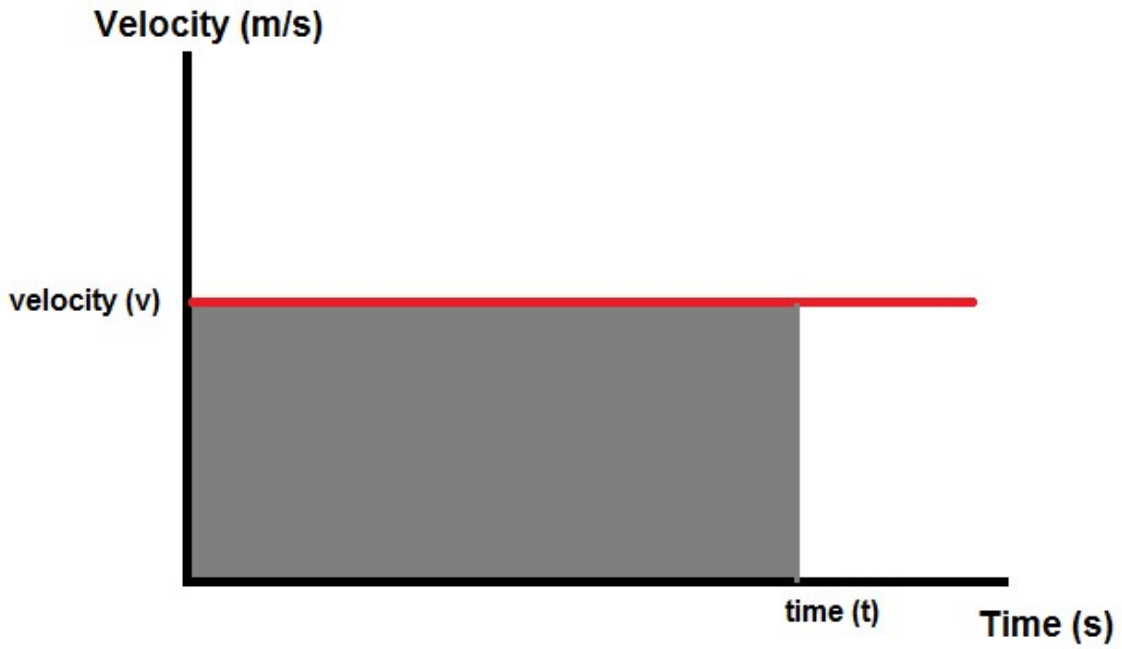
Understanding objects in motion

As mentioned above, two types of motion will be examined, linear and rotational motion. By understanding linear motion, the ability to grasp circular motion will be much easier.

Linear motion

In the most basic sense, an object in motion could be derived from a graph charting uniform velocity against time as demonstrated below:

constant velocity vs time



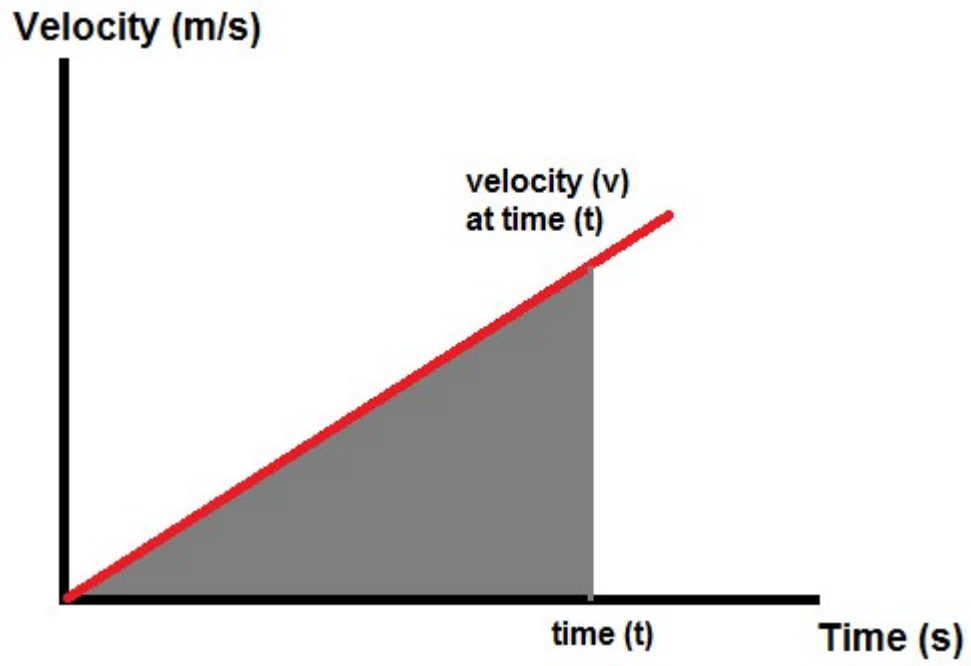
with a constant velocity, the total distance travelled in a certain time period can easily be determined by calculating the area of the rectangle below the velocity line as shown below:

$$s = vt \quad [1]$$

Where:

- s = displacement
- v = velocity
- t = time elapsed

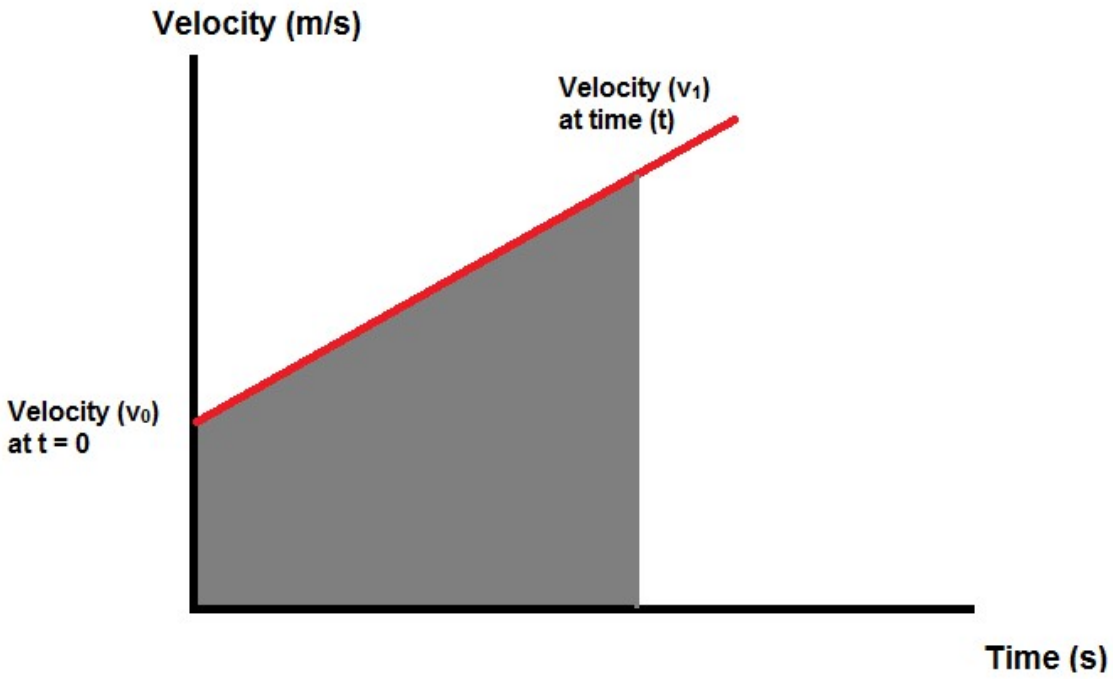
Uniform acceleration vs time



The total displacement(distance in this case) can still be determined by calculating the area of a triangle as shown below:

$$s = \frac{1}{2} v\, t \quad [2]$$

However, if the object is already in motion i.e. v > 0 when t = 0.



Then by finding the change in velocity during the total time elapsed **t**; we can determine the distance travelled during time elapsed **t** again through the equation below

$$s = \frac{1}{2}\ (v_1 - \ v_0)t \quad [3]$$

Where:

- v_0 = the initial velocity of the object
- v_1 = the final velocity of the object

Also through the use of the graph above, we can determine the rate of change of velocity as the gradient of the line. This is known as the acceleration of an object and can be calculated as shown bellow:

$$a = \frac{v_0 - v_1}{t} \quad [4]$$

Where:

- a = acceleration of the object

Further equations of motions can also be derived from an acceleration vs time graph as shown below

(draw constant acceleration/time graph here)

As demonstrated earlier with the constant velocity against time diagram, we can calculate the velocity at the total time elapsed by calculating the area of the rectangle under the line as demonstrated below

$$v = a\ t \quad [5]$$

And the same approach can be used when calculating the current velocity with a constantly increasing acceleration by calculating the area of the triangle instead of a rectangle as shown below

(draw constantly increasing acceleration/time graph here)

$$v = \frac{1}{2}\ a\ t$$

If v > 0 when t = 0

$$s = u + \frac{1}{2} a\ t$$

By substituting the equation for velocity as shown in eq.1 we get the following

$$\frac{s}{t} = u + \frac{1}{2} at$$

And re-arrange to get an equation for the displacement s

$$s = ut + \frac{1}{2}\ at^2 \quad [7]$$

Non uniform displacement, velocity and acceleration

So far, all equations where derived from uniform displacement, velocity and acceleration. However, in realistic scenarios, the plotting of velocity against time would look something like this.

(insert non-uniform velocity / time)

Therefore, more approaches are required to determine the displacement or acceleration. One method would be to divide the graph up into segments

Segmentation and the trapezium rule

One method of determining values for either displacement, velocity or acceleration would be to segment the graph into multiple segments and then calculate the average values from those segments as demonstrated below.

(Demonstrate using the trapezium rule on the non-uniform velocity time graph)

In order to determine the acceleration at a given segment, we simply apply the same calculation used with the uniform velocity/time graph as shown in eq. 4 shown below

$$a = \frac{v_1 - v_0}{t}$$

Therefore, if we want to assess the average acceleration across the velocity/time graph, we could get the mean value of all acceleration segments as shown below

$$a = \frac{\left(\left(\frac{v_1 - v_0}{t_0}\right) + \left(\frac{v_2 - v_1}{t_1}\right) + \left(\frac{v_1 - v_0}{t_2}\right) + \cdots \left(\frac{v_{n+1} - v_n}{t_n}\right)\right)}{n}$$

Where:
n = number of segments

Also using this method of segmentation would also allow us to more accurately determine the area under the velocity curve to get a more accurate value of displacement through the use of the trapezium rule as demonstrated below.

$$s = \frac{1}{2}\, t\, [(v_0 + v_n) + 2(v_1 + \ v_2 + \cdots + v_{n-1})]$$

And with a variable acceleration/time graph

(Insert variable acceleration/time here)

As demonstrated with determining the displacement using the trapezium rule, we can also determine the velocity using the trapezium rule as shown below

$$v = \frac{1}{2}\, t\, [(a_0 + a_n) + 2(a_1 + \ a_2 + \cdots + a_{n-1})]$$

However, the values obtained through segmentation will never be 100% accurate. However, by introducing more segments to the graph, the accuracy of the value obtained will get much closer to the actual value.

Using differential and integral equations for non-uniform motion

As demonstrated with segmentation, the more segments in the graph, the more accurate the final result. Therefore, through observing each segments. It could be determined that velocity is a measure of the rate of change of displacement against the change in time as shown below

(insert variable velocity/time graph)

And that acceleration is a measure of the rate of change of velocity against the change in time as demonstrated below as well

(insert variable acceleration/time graph)

Therefore, we can determine that this will remain the case with an infinitely small value for t as shown bellow

Therefore allowing for the use of differential equations as shown from the graphs above, velocity is the differential of displacement as acceleration is the differential of velocity.

where:

- v = the velocity(speed with a vector)of the object in motion
- S_0 = the initial displacement (position) of the object
- S_1 = the final displacement of the object (think of $s_0 – s_1$ as the distance travelled)
- t_0 = the initial time of motion
- t_1 = the final time of motion (think of $t_1 – t_0$ as the total time elapsed

Therefore, from the equation above, It can be said that velocity is a measure of the change in displacement (distance travelled) against the change in time. Therefore to calculate the velocity at a given instance; we can take (s_0 - s_1) to be Δs and and ($t_0 – t_1$) to be Δt and can be written as followed:

$$\mathbf{v = \Delta s/\Delta t}$$

where:

- Δ = the change (so Δs is the change in displacement or distance travelled)

Now velocity has been expressed with the change of displacement against the change in time, the equation above can be written as a differential equation for the changing **Δ** for **d** where we take the time elapsed as a value infinitely small as shown below:

$$\boldsymbol{v = \frac{ds}{dt}}$$

The equation above does not account for the acceleration of the object. However, acceleration is simply the change of velocity over a given period of time. Therefore, the equation for acceleration can be written as the following:

$$\mathbf{a = v_1/t_1 – v_0/t_0}$$

Where:

- v_1 = the final value for velocity
- v_0 = the initial value for velocity
- a = rate of change of velocity over time period ($t_0 – t_1$)

Therefore, Like the equation for velocity, the equation for acceleration over a change in velocity and time can be denoted as the following:

$$\mathbf{a = \Delta v/\Delta t}$$

And for a given instance where **t** is infinitely small of acceleration:

$$\mathbf{a = dv/dt}$$

Therefore, it can be said that velocity is a differential equation of displacement and acceleration is a differential equation of velocity. Therefore, acceleration can be viewed as the second differential of displacement as demonstrated below:

$$\mathbf{A = d^2s/dt^2}$$

As we have derived equations from displacement to velocity to acceleration. The same can be done in reverse through integrating acceleration to get the value for velocity and integrating velocity to get the value for displacement as shown below:

$$v = \int a\, dt$$

And

$$s = \int v\, dt$$

This can be applied whether the acceleration of an object is constant or is constantly varying with respect to time for example:
if an acceleration profile takes the value; $a = vt^2 + 2t$, then the velocity would be calculated as

$$v = \int vt^2 + 2t\ dt$$

Which would give us $v = \frac{vt^3}{3} + t^2$

Circular motion

(implement similar equations as shown above to derive equations for circular motion and how just by changing the radius of an object can change the speed in which a point on the circumference of the circle will be spinning)

In order to understand Circular Motion, one must be familiar with Circular Geometry.

Circular geometry

Expressions for calculating the geometry of a circle can be derived from the following drawing of a circle

Collisions between objects

(implement derivations of equations of momentum i.e. conservation of momentum, impulse and collisions with friction)

Implementing real life equation into code

In order to express kinematic equations into code, it must be understood that a typical computer has no notion of how physics work in real life. Therefore every kinematic behaviour must be explicitly coded into the library.

Time, the most important variable

As demonstrated by the equations of motions above. The most important variable for calculating either the location, velocity or even acceleration would be time as the change in time would ultimately result in a change in displacement of an object in motion or a change in velocity for an object accelerating. Therefore, in order for everything to function as it should, time is to be considered the first variable to implement.

Location, using vectors to simulate the location of an object

In order for the computer to register the concept of a location, everything must be coded as a vector. Therefore it could state that the location of an object can be denoted by its vectors i and j (and also k for 3d vectors). For 2d vectors, the computer can take the vector co-ordinates as the location on the page depending on the size of a screen, determining the location of an object much like a navigator would do using the longitude and latitude of a map. Therefore, a uniform mesh would be required in order to create a grid to detail where the object would be at any given instance of time

Linear Motion and direction, using vectors to simulate motion

Unlike real life where speed (motion without direction) is easier to calculate that velocity (motion with direction) in a computer however, it is practically impossible to simulate motion without the use of vectors.

Therefore the total velocity of an object can be determined by using Pythagoras theorem

$$magnitude\ of\ velocity = \sqrt{i^2 + j^2}$$

And the direction could be determined through the use of trig equations

$$angle\ of\ travel = tan^{-1}\left(\frac{j}{i}\right)$$

Circular motion using matrices to simulate circular motion and the rotation of an object.

As computers lack the ability to physically rotate an object. Therefore, in order to simulate circular motion, the use of matrices is essential for simulating the rotation of an object… to be continued

Production schedule

Incorporate a Gantt chart to outline the plan of action in how this project will be completed.

Todo:

1. Complete circular motions by incorporating these equations and how they are derived (note all calculations are done in radians)
   angle rotated $\Delta\vartheta = \frac{\Delta s}{r}$

   angular rotation $\omega = \frac{\Delta\vartheta}{\Delta t}$

   linear velocity (speed in which the circumference is spinning) $v = r\ \omega$

   and show further derivations much like we donein linear velocity

2. Implement equations of momentum to demonstrate momentum
   e.g.
   momentum $p = m_1 v_1$
   conservation of momentum $p_1 = p_2$
   momentum of multiple particles = $\sum_n m_n\ v_n$

3. Expand greatly on matrix algebra and demonstrate how that is used to simulate the rotation of a given object through matrix multiplication

4. Create a Gantt chart to provide some basis of a structured approach on tackling this project.

5. Expand on the road map to incorporate how each property will be adapted to the bouncing bomb operation.

References:

1. Physics for Game Developers, 2nd Edition by Bryan Bywalec; David M Bourg (viewed from https://learning.oreilly.com/library/view/physics-for-game/9781449361037/ch02.html)

2. A summarised comparison of kotlin vs java https://www.konstantinfo.com/blog/kotlin-vs-java/

3. History of the bouncing bomb operation http://www.dambusters.org.uk/the-dam-raids/the-bomb/the-bouncing-bomb/

4. How the bouncing bomb worked http://www.chm.bris.ac.uk/webprojects2001/moorcraft/The%20Bouncing%20Bomb.htm

5. Circular motion http://web.mit.edu/8.01t/www/materials/modules/chapter06.pdf

6. Relation of rotation vector and angular velocity https://physics.stackexchange.com/questions/433102/relation-between-rotation-vector-derivative-and-angular-velocity-when-the-rotati?rq=1

7. Circular motion equaitons https://www.khanacademy.org/science/ap-physics-1/ap-centripetal-force-and-gravitation/introduction-to-uniform-circular-motion-ap/a/circular-motion-basics-ap1

8. Physical simulations computed https://www.falstad.com/mathphysics.html

9. https://gamedevelopment.tutsplus.com/tutorials/how-to-create-a-custom-2d-physics-engine-the-basics-and-impulse-resolution--gamedev-6331

10. https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics

11. https://pet.timetocode.org/

ʊ