My approach to the BPhO computational challenge

Moses Bejon

August 7, 2024

Contents

Abstract

I decided to solve the mathematical problems of this challenge myself, rather than simply implementing the worked solutions provided to us. I have written this to detail how I came to the answers I came to and go over the mathematical basis underlying the operations of my web app.

1 Setting up my environment

1.1 Structure

I picked JavaScript as my tool for the challenge due to its portability to different platforms and the ability to host it online. I built it as a single page application, as this allows it to act more like an app than a website.

1.2 Graph plotting

I decided to create a plot class, which would plot data though an SVG. I felt building the tool myself gave me more flexibility over what it looked like, which allowed me to implement complex interactions like animations later on down the line. The class worked individually from any specific task, calculated data is passed into the class as an array of points[1] and the class plots it. It also has the option to plot labelled points. I then gave the graph class methods for determining an appropriate scale using the maximum and minimum values of the graph, to ensure the entire contents of the dataset would fit onto the graph.

One interesting problem I ran into was how to choose a scale that a user would find reasonable, as dividing my scaled graph into equal proportions yields long displeasing decimals. In the end, I came up with the following formula for determining elegant steps to go up in (where x is the top of the graph):

$$10^{\lfloor \log_{10} x \rfloor - 1}$$

This formula picks reasonable values easily expressible in base 10. However, upon initial testing, I noticed a key issue. For some values of x, the step sizes were very small, and thus the axes became filled up with numbers and cluttered. In order to counteract this, I added a check to see how many points there are[2] and then, depending on this value, removing a proportion of these axes values at regular intervals (I decided to keep the extra grid-lines).

2 Task 1

For the first task, I had to take inputs:
-launch angle
-strength of gravity
-launch speed
-launch height

And output a graph of x position against y position (neglecting drag) using discrete, fixed, time step intervals. I decided to also add the time step as an input, allowing users to experiment with higher and lower resolution graphs. I began by finding the components of the initial velocity using the angle.

$$v_x = u\cos(\theta)$$
$$v_y = u\sin(\theta)$$

Since there are no forces acting in the horizontal direction this velocity will remain constant, and we can conclude that at point t in time:

$$x = v_x \cdot t$$

[1] Each point is stored as its own array of 2 values.

[2] This was done by dividing the top by the step size.

In the y direction, the projectile is accelerating downward at the rate g. Using constant acceleration formulae:

$$s = u \cdot t + \frac{1}{2} \cdot a \cdot t^2$$
$$y = v_y \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

The above formula gives the distance change in y due to its velocity and acceleration. However, since y initially begins at height h:

$$y = h + v_y \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

3 Task 2

3.1 Defining y in terms of x

For task 2 I required a function of y in terms of x. As I already had both y and x in terms of t, I could rearrange my original formulae in order to attain this:

$$t = \frac{x}{v_x}$$
$$y = h + v_y \cdot \frac{x}{v_x} - \frac{1}{2} \cdot g \cdot \left(\frac{x}{v_x}\right)^2$$
$$y = h + \frac{v_y}{v_x} \cdot x - \frac{g}{2v_x^2} \cdot x^2$$

3.2 Plotting the graph

Having y in terms of x, I now needed to plot the values of y for an equally spaced array of x values. In order to determine appropriate spacing, I decided to use the length of my graph divided by 100, so each plot would have 100 points on it. In order to find the end of the graph, I needed to attain the value of x where y is equal to 0:

$$0 = h + \frac{v_y}{v_x} \cdot x - \frac{g}{2v_x^2} \cdot x^2$$

Using the quadratic formula I was able to find:

$$x = \frac{\frac{v_y}{v_x} \pm \sqrt{\left(\frac{v_y}{v_x}\right)^2 + \frac{2gh}{v_x^2}}}{\frac{g}{v_x^2}}$$

As we are looking for the greater value of x:

$$x = \frac{\frac{v_y}{v_x} + \sqrt{\left(\frac{v_y}{v_x}\right)^2 + \frac{2gh}{v_x^2}}}{\frac{g}{v_x^2}}$$
$$x = \frac{v_x^2}{g} \cdot \left(\frac{v_y}{v_x} + \frac{1}{v_x}\sqrt{v_y^2 + 2gh}\right)$$
$$x = \frac{v_x}{g} \cdot \left(v_y + \sqrt{v_y^2 + 2gh}\right)$$

This value of x could then be divided by 100 and a point could be plotted for each value.

3.3 Finding the apogee

Being able to now plot the graph, the last thing I needed was to find the apogee. As the apogee's location would be at the same point as the turning point of the graph, we are looking for the point where $\frac{dy}{dx} = 0$:

$$\frac{dy}{dx} = \frac{v_y}{v_x} - 2 \cdot \frac{g}{2v_x^2} \cdot x = 0$$
$$v_y - \frac{g}{v_x} \cdot x = 0$$
$$x = \frac{v_y \cdot v_x}{g}$$

In order to find the y coordinate of this point, we can substitute:

$$y = h + \frac{v_y}{v_x} \cdot \left(\frac{v_y \cdot v_x}{g}\right) - \frac{g}{2v_x^2} \cdot \left(\frac{v_y \cdot v_x}{g}\right)^2$$

$$y = h + \frac{v_y^2}{g} - \frac{v_y^2}{2g}$$

$$y = h + \frac{v_y^2}{2g}$$

4 Task 3

4.1 Finding θ to pass through target

For task 3 I had some point, (X, Y), that the trajectory had to move through being launched from position (0,0) and I had to find the two angles that could achieve this. This meant I had to work in terms of $u\cos(\theta)$ and $u\sin(\theta)$ in order to find the angle. The working shown in appendix A shows that:

$$\tan(\theta) = \frac{u^2 \pm \sqrt{u^4 - 2u^2gY - g^2X^2}}{gX}$$

4.2 Finding minimum launch velocity

At the minimum value of u there is no "low ball" or "high ball", as in these scenarios we are wasting our velocity. Since there is only one solution for θ, we can conclude the discriminant is equal to zero:

$$u^4 - 2gYu^2 - g^2X^2 = 0$$

$$u^2 = \frac{2gY \pm \sqrt{4g^2Y^2 + 4g^2X^2}}{2}$$

$$u = \pm\sqrt{gY \pm g\sqrt{Y^2 + X^2}}$$

As our initial speed cannot be negative:

$$u = \sqrt{gY \pm g\sqrt{Y^2 + X^2}}$$

As X and Y are both greater than zero:

$$Y < \sqrt{Y^2 + X^2}$$

$$gY < g\sqrt{Y^2 + X^2}$$

$$gY - g\sqrt{Y^2 + X^2} < 0$$

Re-examining our equation for u, if we use the negative sign we will not get a real solution, therefore:

$$u = \sqrt{gY + g\sqrt{Y^2 + X^2}}$$

In order to find the corresponding value of θ, knowing our discriminant is zero:

$$\tan(\theta) = \frac{u^2}{gX}$$

$$\tan(\theta) = \frac{\left(\sqrt{gY + g\sqrt{Y^2 + X^2}}\right)^2}{gX}$$

$$\tan(\theta) = \frac{Y + \sqrt{Y^2 + X^2}}{X}$$

5 Task 4

For task 4 I had to find the horizontal distance travelled. This is the positive value of x when y is equal to zero. From workings seen at appendix A it should be clear that:

$$2u^2y = 2u^2h + 2u^2x\tan(\theta) - gx^2 - gx^2\tan^2(\theta)$$

Substituting y = 0:

$$2u^2h + 2u^2x\tan(\theta) - gx^2 - gx^2\tan^2(\theta) = 0$$

In order to find the maximum distance, we need to find the maximum value of x while θ

varies. We therefore need to find the point where the rate of change of x is zero with respect to θ, or:

$$\frac{dx}{d\theta} = 0$$

Using the implicit differentiation shown at appendix B, we can show that the greatest horizontal distance, and the angle at which this occurs, is given by:

$$x = \frac{u\sqrt{2hg + u^2}}{g}$$
$$\tan(\theta) = \frac{u}{\sqrt{2hg + u^2}}$$

6 Task 5

For task 5 I needed to find the bounding parabola of all possible points (X, Y) reachable by the given trajectory and the given launch speed as the angle θ varies. Knowing:

$$\tan(\theta) = \frac{u^2 \pm \sqrt{u^4 - 2u^2gY - g^2X^2}}{gX}$$

from appendix A. In order for the angle θ to be a real number, the discriminant must be greater than or equal to zero. Therefore:

$$u^4 - 2u^2gY - g^2X^2 \geq 0$$

Therefore the bounding parabola follows the equation:

$$u^4 - 2u^2gY - g^2X^2 = 0$$

Making Y the subject here in order to calculate from X values as we go along the graph:

$$2u^2gY = u^4 - g^2X^2$$
$$Y = \frac{u^4 - g^2X^2}{2u^2g}$$

7 Task 6

For task 6 I needed to calculate the length of my trajectories. The length of a function between x values a and b can be given by:

$$\int_a^b \sqrt{1 + [f'(x)]^2}$$

Here, a will equal 0, as the trajectory starts at x=0, and b will be the x value found at 3.2. The function can be replaced with the definition of y in terms of x at 3.1. All that is left to do is to differentiate, then integrate:

$$\frac{dy}{dx} = \frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot x$$
$$\int_0^X \sqrt{1 + \left(\frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot x\right)^2} dx$$

See appendix C for the solution to this integral.

8 Task 7

Task 7 required me to plot the distance from the origin against time and find any turning points. Using the time equations derived at 2:

$$D = \sqrt{(v_x \cdot t)^2 + (h + v_y \cdot t - \frac{1}{2}g \cdot t^2)^2}$$

See differentiation and turning points at appendix E. Once the t values at the turning points are calculated, x, y and D can be calculated as:

$$x = v_x \cdot t$$
$$y = h + v_y \cdot t - \frac{1}{2}g \cdot t^2$$
$$D = \sqrt{x^2 + y^2}$$

9 Task 8

9.1 Computing the change in position over a given time

Task 8 required another numerical approach. This time, to incorporate a bounce using some coefficient of restitution to determine what proportion of the projectile's vertical speed is maintained after a bounce[3]. The numerical approach I chose was using discrete time steps. The projectile's x position after some time period can be given by:

$$x' = x + v_x t$$

The projectile has constant acceleration in the y direction due to gravity, so:

$$v'_y = v_y - gt$$

In order to find the projectile's y position after some time period we can integrate the right-hand side as the integral of velocity is displacement.

$$y' = \int (v_y - gt)\, dt$$
$$y' = v_y t - \frac{1}{2} g t^2 + c$$

When t is zero, y has not moved from its original position, as you cannot move any distance in zero seconds. Therefore, c here is equal to y's original position. Therefore:

$$y' = y + v_y t - \frac{1}{2} g t^2$$

[3]The horizontal speed will not be taken into account here as it remains constant

9.2 Computing the result of a bounce

y' should not be able to be lower than zero, and if it is computed to be such by the above formula, a bounce must be computed. In order to find the speed of the projectile bounces, the speed of the projectile at the moment it collides with the ground must be found. When the projectile collides with the ground, y = 0. Therefore, we need to find t such that:

$$0 = y + v_y t - \frac{1}{2} g t^2$$

Using the quadratic formula:

$$t = \frac{-v_y \pm \sqrt{v_y^2 - 4\left(-\frac{1}{2}g\right)(y)}}{2\left(-\frac{1}{2}g\right)}$$
$$t = \frac{v_y \pm \sqrt{v_y^2 + 2g(y)}}{g}$$

g and y are both positive in the above expression, and they are being added to the square of velocity before it is being square rooted. This means that the value being added/subtracted from velocity is greater than velocity. However, if it were being subtracted we would attain a negative time value, therefore it must be being added.

$$t = \frac{v_y + \sqrt{v_y^2 + 2gy}}{g}$$

And the velocity immediately before the collision is given by $v_y - gt$. The y velocity after the collision is, therefore:

$$C \cdot -(v_y - gt)$$

Where C is the coefficient of restitution. As t in this case is only a proportion of the time

step, we can then repeat the entirety of the above formulae to compute what happens in the rest of the time step.

In addition, when a bounce is completed, a counter for the number of bounces is incremented and checked against the total number of allowed bounces by the user. If this number is greater than the total number of allowed bounces, the graph is complete.

10 Task 9

Using the Verlet method in discrete time steps to take into account air resistance (which is assumed to vary with the square of velocity):

$$\vec{a} = -k|\vec{v}|\vec{v} - \begin{bmatrix}0\\g\end{bmatrix}$$
$$\vec{r}\,' = \vec{r} + \vec{v}t + \frac{1}{2}\vec{a}t^2$$
$$\vec{v}\,' = \vec{v} + \vec{a}t$$

where:

$$k = \frac{c_D \rho A}{2m}$$

where c_D is the drag coefficient, ρ is the air density and A is the cross-sectional area.

These equations come from the "detailed BPhO tasks" PDF, which I had avoided using so far, but for this task I was unsure of how to take air resistance into account in two dimensions. I made some minor modifications to the acceleration equations (for reasons of computational efficiency) but they still follow the original equations very closely[4].

11 N-dimensional projectile extension

11.1 Introduction

I noticed that the formula I had used to compute the motion of projectiles in task 9 was entirely in terms of vectors. This meant it was trivial to write algorithms to compute the positions of particles in any number of dimensions. I will be using g as an n-dimensional vector here, as the alternative of it being in a single direction would not result in interesting simulations. As such, the equations from task 9 are slightly modified as shown:

$$\vec{a} = -k|\vec{v}|\vec{v} + \vec{g}$$
$$\vec{r}\,' = \vec{r} + \vec{v}t + \frac{1}{2}\vec{a}t^2$$
$$\vec{v}\,' = \vec{v} + \vec{a}t$$

One of the key challenges here was displaying my data. I settled for a 3D view, though drawing a trajectory line graph, as I had in previous challenges, wouldn't easily depict the projectile's position into the screen (along the z axis). I settled on a ball instead that moved through the region in real time.

As the ball was moving through the scene in real time, I didn't want it to just stop

[4]The main change was that in the original formulae provided $|\vec{v}|^2$ is divided by $|\vec{v}|$. The original form captures what is physically happening in the system but, from a computational efficiency point of view, it makes sense to simplify this to $|\vec{v}|$

when it got to the edge of the screen or keep falling forever, I thought it would be appropriate to include the bounce from task 8. However, I had not yet incorporated the bounce into the air resistance model.

11.2 Incorporating a bounce into the air resistance model

As the ball in now in a box, a bounce should now be computed if any of the entries in the calculated next position vector ($\vec{r}'$) are such that they would make any part of the ball outside the box. Therefore, for a cubic box of side length l and a ball of radius R with position vector $\vec{r}$ (taken at the centre of the ball):

$$\forall r_i, R \leq r_i \leq l - R$$

If the ball is calculated to be in a position such that the above is not true, a bounce should occur. The first step would be to calculate the velocity of the ball at the exact time it collided with the wall in order to accurately calculate the velocity of it after it bounced. For the offending entry, there must be some point where:

$$r_i' = r_i + v_i t + \frac{1}{2} a_i t^2 = b$$

where either:

$$b = R$$
$$b = l - R$$

In accordance to the workings presented in task 8 (9.2):

$$t = \frac{v_i + \sqrt{v_i^2 + 2a_i(r_i - b)}}{a_i}$$

The velocity after this period of time is:

$$\vec{v}' = \vec{v} + \vec{a}t$$

But the velocity after the bounce is:

$$-C(v_i + a_i t)$$

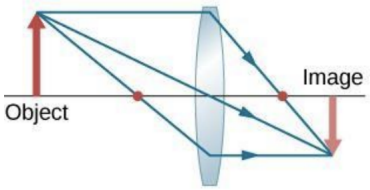
The reason we had to calculate the velocity after this time for each dimension is that when we calculate how the ball will move after the collision (with a new acceleration calculated based on the all the new calculated velocities) this data is required. If multiple collisions occur within a time step, the one with the lowest t will be used to attain the next set of accelerations. Each time there is a collision, the above process is repeated to calculate what happens next, including the collision check.

11.3 Displaying my data in one and two dimensions

In order to display my numbers, I require a different display for each number of dimensions, each with some new part that displays the position of the ball along that new dimension. For 2D, I could use a circle with position indicated by the number for each dimension. For 1D, while I could still use a circle that only moves in one direction rather than 2, I feel this doesn't quite capture the fact that the simulation is one dimensional. The curve of a circle might seem to suggest some kind of asymmetry between the thick parts and thin parts of the circle. I therefore decided to use a rectangle for one dimension.

11.4 Displaying my data in three dimensions

As with typical perspective, the size of the circle will indicate its position in the 3rd dimension. In order to ascertain the exact relationship between the distance of the ball from the camera and its size on the screen, we need to analyse how the converging lens transfers the light of an object onto a screen.



Consider the triangle formed by the tip of the object, the bottom of the object and the optical centre of the lens. Compare it to the triangle formed by the tip of the image, the bottom of the image and the optical centre of the lens. The triangles meet at the optical centre, where two straight lines intersect. This therefore means that the angles at the optical centre of both triangles are equivalent. In addition, as the object and image stand upright, they both share a ninety-degree angle between the principle axis and the object/image. As they share two angles, the triangles are similar. As d and f are similar sides (d being the distance between the lens and the object and f being the focal length) and h and h' are similar (h being the height of the object and h' being the height of the image):

$$h' = \frac{fh}{d}$$

However, it is not enough just to scale down the size of the ball as it gets further away. This is because as the space the ball is confined to moves further away, it must also get smaller. This essentially means we must apply the same formula to the x and y values of the position of the ball. However, plugging x and y into the above formula will result in the ball getting closer and closer to the top left corner (where x and y are equal to zero). We would like the ball to get closer and closer to the centre as it gets further away. What we therefore need to do is find the distance of the ball from the centre and shrink this as shown:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \end{bmatrix} + \frac{f}{d} \begin{bmatrix} x - c_x \\ y - c_y \end{bmatrix}$$

where the centre of the screen is (c_x, c_y).

However, according to these formulae, no matter what focal length I set for my lens, as the distance gets closer to the screen (approaches zero) the size of the ball approaches infinity. This is problematic if we are to display the data on a finitely large computer screen. To overcome this, we must declare the edge of the box to be some finite distance away from the lens. I should also like the ball to make full use of the screen size it is given. As the box's width is l, we would like the following to be true:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \frac{l}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{f}{d} \cdot \frac{l}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} r \\ r \end{bmatrix}$$

$$\begin{bmatrix} l \\ l \end{bmatrix} = \frac{l}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{f}{d} \cdot \frac{l}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} r \\ r \end{bmatrix}$$

In both cases, the full expanse of the screen is used if:

$$\frac{f}{d} = 1 - \frac{2r}{l}$$

I thought a reasonable distance of a box away from a camera might be 1 m.

$$\frac{f}{1} = 1 - \frac{2r}{l}$$
$$f = 1 - \frac{2r}{l}$$

This meant the focal length of my camera had to be dynamically adjusted to the length of the box and radius of the ball as they updated, so this is a pretty expensive camera to say the least.

11.5 Beyond the third dimension

Representing a third dimensional scene on a two-dimensional screen already loses detail. For example, it is ambiguous how much the scale of the ball is based on its radius or its distance from the camera. Luckily, as humans have a lot of expertise with 2D projections of 3D scenes, humans can easily compensate for this. However, humans do not have the same experience projecting 4D scenes onto 2D screens. Therefore, even if I found a sensible way to do this projection, the resulting graphic would likely be meaningless to humans.

I decided, instead, to not project but rather encode extra dimensions within the colour of the room. I felt this helped capture the epileptic chaos that is trying to comprehend the fourth dimension. I decided to use the three colour channels red, green and blue. The less red the room, the closer the ball is to you in the fourth dimension. The less green the room, the closer the ball is to you in the fifth dimension, and so on.

12 Projectile launched from earth extension

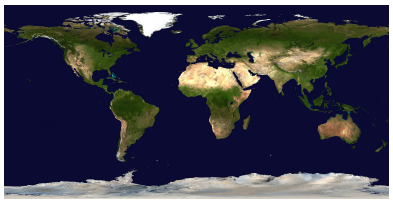
12.1 Earth projections

As JavaScript does not natively support 3D meshes[5], I decided to instead use a 2D map as earth and project this onto a sphere. I chose the equirectangular projection as it had a very simple relationship between geographic coordinates (latitude and longitude) with map pixel coordinates (x,y). The relationship is as follows:

$$x = W \cdot \frac{\varphi^\circ + 180}{360}$$
$$y = H \cdot \frac{-\lambda^\circ + 90}{180}$$

where W and H are the width and height of the map and φ is the longitude and λ is the latitude.

Here is the equirectangular projection of earth:



12.2 Ray casting

What I intend to do is only ray casting in the most fundamental sense, that the

[5]Whilst there are a few CSS 3D options, like a perspective attribute, there is no easy way to import a whole mesh file into a webpage (such as a .obj) without the use of external libraries (like three.js). These libraries, however, were large and clunky, which meant they took a while (couple of seconds) to load up. I knew a bespoke solution would take negligible time to load and so decided to do that.

camera emits a ray and the ray's first intersection is what is rendered to the camera. Modern ray tracing algorithms, upon finding this first intersection, will approximate the solution of an integral in order to determine what colour the pixel should be. This is in order to take lighting into account. I will simply be using whatever colour the pixel on my map is. This means that my algorithm will not consider any lighting[6]. Each ray should pass through the focal point of the camera and the location of each pixel on the camera.

The earth, for convenience, will be centred at (0,0,0). This means, assuming the earth is a sphere with radius r, the equation of its surface is:

$$x^2 + y^2 + z^2 = r^2$$

The camera's position will be defined in terms of its focal point by distance d, its distance from the centre of the earth. This makes the focal point of the camera (d,0,0).[7] This means the centre of the screen of the camera is located at ($d - f$,0,0) where f is the focal length of the camera.

The camera also has a sensor size. This indicates the width/height of the camera's screen. This will be indicated by S. The corner's of the camera's screen can therefore be found to be:

$$(d - f, \frac{S}{2}, \frac{S}{2})$$
$$(d - f, -\frac{S}{2}, \frac{S}{2})$$
$$(d - f, \frac{S}{2}, -\frac{S}{2})$$
$$(d - f, -\frac{S}{2}, -\frac{S}{2})$$

In addition, each pixel's point can be defined as the point[8]:

$$\vec{p} = (d - f, -\frac{S}{2} + \frac{Si}{n} + \frac{S}{2n}, -\frac{S}{2} + \frac{Sj}{n} + \frac{S}{2n})$$
$$\vec{p} = (d - f, \frac{S(2i + 1 - n)}{2n}, \frac{S(2j + 1 - n)}{2n})$$

for an image of resolution n for pixel (i,j). A pixel will from now on be referred to as $\vec{p}$.

We then must find the ray (line) that intersects the points $\vec{p}$ and $(d, 0, 0)$ (the focal point). A line can be defined by two vectors, one being a point on the line ($\vec{c}$) and the other being the direction the line moves ($\vec{m}$):

$$\vec{l} = \vec{m}t + \vec{c}$$

Where $\vec{l}$ is any point on the line and t is any real number. In this case, $\vec{c}$ can be any of the two points but, for computational efficiency, I have selected $(d, 0, 0)$[9]. $\vec{m}$ is the

[6]Which is why I have referred to it as ray casting rather than ray tracing. Ray tracing is commonly associated with lighting.

[7]The axis along which the camera's position is not zero is chosen arbitrarily.

[8]The extra $\frac{S}{2n}$ term comes from the fact that the point that will be used to generate a pixel's colour should be the centre of that pixel

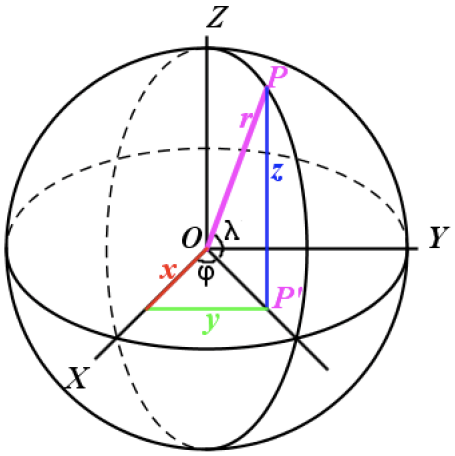
[9]This is because the two entries that are zero do not need to be added, as adding zero to a number does not affect the number, meaning two fewer calculations are required per pixel.

difference between these two points. Therefore[10]:

$$\vec{l} = \left(\vec{p} - \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}\right) t + \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}$$

The next step is to find the intersection points between the line and the sphere. The intersection is found at appendix G in terms of x, y and z. However, these coordinates need to be in terms of latitude and longitude in order to use the projection formulae presented in 12.1.

The following diagram shows this relationship:



Consider the right-angled triangle with adjacent and opposite sides x and y and angle φ.

$$\varphi = \arctan\left(\frac{y}{x}\right)$$

Consider the right-angled triangle with opposite side z, the hypotenuse of the previous triangle and angle λ.

$$\lambda = \arctan\left(\frac{z}{\sqrt{x^2 + y^2}}\right)$$

12.3 Handling rotations

Currently, our camera is statically positioned. Moving the camera and the position of every pixel on its screen would not only require me to do a huge amount more maths, but would also require every single ray that has already been cast to be cast all over again (being an incredibly computationally expensive solution only usable on very fast machines).

Instead, if we rotate the texture of the earth around the earth we don't have to recast a single array, only check the new pixel underneath the ray.

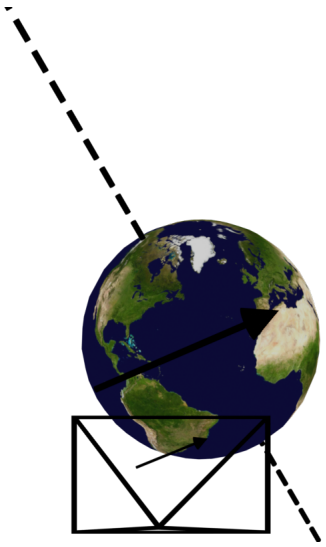
Recall the formulae presented in 12.1. One might think if we simply add some offset to φ we will have rotated the sphere by the offset.

$$x = W \cdot \frac{(\varphi^\circ + \varphi_0^\circ) + 180}{360}$$

However, this method only works while the camera is parallel to the equator and the user is rotating along the x direction of their screen[11]. When one rotates a 3D view, what they expect to happen is reasonably complicated. It is shown in the following diagram:

[10] I have opted to subtract $(d, 0, 0)$ from $\vec{p}$ as it allows t to be positive.

[11] This failed algorithm for rotation enabled me to easily rotate the earth about its own axis for any angle, which was useful when I had to consider the earth's rotation over time of 7.2921150×10^{-5} radians per second.



The black box is the camera. When the user clicks and moves their mouse along the arrow, that arrow gets projected onto the globe. An axis is drawn perpendicular to the arrow and parallel to the camera, and an angle proportional to the length of the arrow is used to rotate the earth about that axis.

I decided to use a matrix transformation on the (x, y, z) position of each point before I convert them into λ and φ.[12] This allowed me to use the general 3D rotation matrix, defined in terms of a unitary axis vector. The full matrix is given at I. In order to find the correct unitary vector, I started with the vector the user had drawn onto the screen[13]:

$$\begin{bmatrix} 0 \\ u_y \\ u_z \end{bmatrix}$$

This vector, as is, would only be valid at the very beginning of the simulation when the user is facing the earth along the x-axis. In order to transform this vector into the axis of rotation, we must move it so that it is parallel to the camera's actual position. One way to achieve this is to keep track of the current overall rotation matrix of the scene (that is, a matrix that takes into account every single rotation performed by the user so far). This matrix starts off as the identity matrix and every time a rotation occurs this rotation matrix gets multiplied on to the current overall rotation matrix. This way, this matrix captures the entire change of position to the scene.

12.4 Calculating the points on the trajectory

In order to calculate the points along the trajectory, I can use similar numerical approaches I have used previously. However, instead of $\vec{g}$ being a constant vector set by the user, it will now be calculated using Newton's law of universal gravitation, which states that:

$$F = G\frac{m_1 m_2}{r^2}$$

I will use subscript p to refer to the projectile and subscript E to refer to the earth.

[12] To make this section as accessible as possible, see appendix H for a breakdown of what matrices are and how they transform vectors.

[13] The reason that the user the vector draws onto the screen has x entry 0 is that the camera is facing in the x direction. The user cannot draw onto the x-axis as it is perpendicular to them and the mouse cannot move through the 3rd dimension.

Let $\vec{r}$ be the vector from the projectile to the centre of the earth and $\hat{r}$ be the unit vector of this vector.

$$\vec{F_p} = G\frac{m_p m_E}{|\vec{r}|^2}\hat{r}$$
$$m_p\vec{a_p} = G\frac{m_p m_E}{|\vec{r}|^3}\vec{r}$$
$$\vec{a_p} = G\frac{m_E}{|\vec{r}|^3}\vec{r}$$

The acceleration of the earth due to attraction to the projectile will be assumed to be negligible.

12.5 Projecting the trajectory onto the camera

In order to project the trajectory onto the camera, we can project each individual point of the trajectory onto the camera and then connect the points with lines. In order to project the point, we can draw a line through the focal point of the camera and the point on the trajectory and the point at which this line intersects the camera's sensor is its projected point.

If we are looking at the point $\vec{r}$ the line connecting it to the focal point is:

$$\vec{l} = \left(\vec{r} - \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}\right) t + \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}$$

In order to find where this line hits the sensor, we need to find at which point the x coordinate is $d - f$ (this is the x coordinate of the screen). Isolating just the x-axis:

$$(r_0 - d)t + d = d - f$$
$$t = \frac{f}{d - r_0}$$

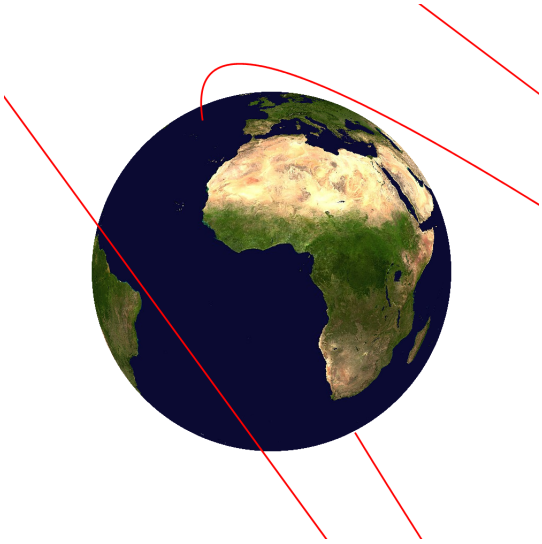
We can substitute to find the y and z coordinates we are interested in.

$$y = \frac{r_1 f}{d - r_0}$$
$$z = \frac{r_2 f}{d - r_0}$$

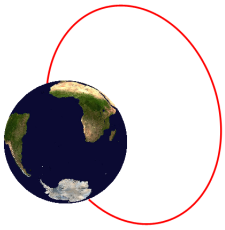
However, this will plot every single point in the trajectory, even if that point is behind the planet. In order to ensure that the point is only plotted if the trajectory is in front of the planet, we must ensure that the t value of the point[14] is less than the t value of the planet.

We also need to ensure that the trajectory point we are projecting is not behind the camera. This was not a problem with the earth as the camera is always pointing towards the earth and never away from it, however, there are scenarios where the trajectory is high enough that they go over the camera, in which case the camera erroneously creates some interesting, yet undesired, artefacts. This is an example of such an artifact:

[14] This will always be 1 as when $t = 1$ is substituted into the equation for $\vec{l}$ the output will always be $\vec{r}$.



When the camera is zoomed out such that it is above the trajectory the artifact goes away:



As the camera's position is fixed, and the world is the one that is rotating, the only way a point can be behind the camera is if its x value is greater than that of the screen. This is a simple check to make.

A Finding θ in terms of X, Y, u and g

$$y = h + \frac{v_y}{v_x} \cdot x - \frac{g}{2v_x^2} \cdot x^2$$
$$Y = 0 + \frac{u\sin(\theta)}{u\cos(\theta)} \cdot X - \frac{g}{2(u\cos(\theta))^2} \cdot X^2$$
$$Y = \frac{X\sin(\theta)}{\cos(\theta)} - \frac{gX^2}{2u^2\cos^2(\theta)}$$
$$Y = X\tan\theta - \frac{gX^2}{2u^2 \cdot \frac{1}{1+\tan^2(\theta)}}$$
$$Y = X\tan\theta - \frac{gX^2(1+\tan^2(\theta))}{2u^2}$$
$$2u^2Y = 2u^2X\tan(\theta) - gX^2 - gX^2\tan^2(\theta)$$
$$gX^2\tan^2(\theta) - 2u^2X\tan(\theta) + 2u^2Y + gX^2 = 0$$
$$\tan(\theta) = \frac{2u^2X \pm \sqrt{4u^4X^2 - 4gX^2(2u^2Y + gX^2)}}{2gX^2}$$
$$\tan(\theta) = \frac{2u^2X \pm 2X\sqrt{u^4 - g(2u^2Y + gX^2)}}{2gX^2}$$
$$\tan(\theta) = \frac{u^2 \pm \sqrt{u^4 - g(2u^2Y + gX^2)}}{gX}$$
$$\tan(\theta) = \frac{u^2 \pm \sqrt{u^4 - 2u^2gY - g^2X^2}}{gX}$$

B Finding θ for the greatest horizontal distance

$$2u^2h + 2u^2x\tan(\theta) - gx^2 - gx^2\tan^2(\theta) = 0$$
$$2u^2x\sec^2(\theta) + 2u^2\frac{dx}{d\theta}\tan(\theta) - 2gx\frac{dx}{d\theta} - 2gx^2\sec^2(\theta)\tan(\theta) - 2gx\frac{dx}{d\theta}\tan^2(\theta) = 0$$
$$2u^2x\sec^2(\theta) + 2u^2(0)\tan(\theta) - 2gx(0) - 2gx^2\sec^2(\theta)\tan(\theta) - 2gx(0)\tan^2(\theta) = 0$$
$$2u^2x\sec^2(\theta) - 2gx^2\sec^2(\theta)\tan(\theta) = 0$$
$$2x\sec^2(\theta)(u^2 - gx\tan(\theta)) = 0$$

x cannot equal zero as x is greater than 0 and $\sec^2(\theta)$ cannot equal zero as in:

$$\frac{1}{\cos^2(\theta)} = 0$$

no value of $\cos^2(\theta)$ can satisfy the equation. Therefore:

$$u^2 - gx\tan(\theta) = 0$$
$$\tan(\theta) = \frac{u^2}{gx}$$

In order to find the point on the graph where the gradient is zero, we find the intersection point between our gradient being zero equation above and the original function.

$$2u^2h + 2u^2x\tan(\theta) - gx^2 - gx^2\tan^2(\theta) = 0$$
$$2u^2h + 2u^2x\left(\frac{u^2}{gx}\right) - gx^2 - gx^2\left(\frac{u^2}{gx}\right)^2 = 0$$
$$2u^2h + \frac{2u^4}{g} - gx^2 - \frac{u^4}{g} = 0$$
$$gx^2 = 2u^2h + \frac{u^4}{g}$$
$$x = \sqrt{\frac{2u^2hg + u^4}{g^2}}$$

x cannot be negative here.

$$x = \frac{\sqrt{2u^2hg + u^4}}{g}$$
$$x = \frac{u\sqrt{2hg + u^2}}{g}$$

In order to find θ:

$$\tan(\theta) = \frac{u^2}{gx}$$
$$\tan(\theta) = \frac{u^2}{g\left(\frac{u\sqrt{2hg+u^2}}{g}\right)}$$
$$\tan(\theta) = \frac{u}{\sqrt{2hg + u^2}}$$

C Solving for the distance by integration

$$\int_0^X \sqrt{1 + \left(\frac{v_y}{v_x} - \frac{g}{v_x^2}\cdot x\right)^2}\,dx$$

Let $u = \frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot x$

$$\frac{du}{dx} = -\frac{g}{v_x^2}$$
$$dx = -\frac{v_x^2 du}{g}$$

Making appropriate substitutions:

$$\int_{\frac{v_y}{v_x}}^{\frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot X} \sqrt{1+u^2} \cdot -\frac{v_x^2}{g} du$$

let $u = \tan(\theta)$

$$\frac{du}{d\theta} = \sec^2(\theta)$$
$$du = \sec^2(\theta) d\theta$$

Making a second set of appropriate substitutions:

$$\int_{\tan^{-1}(\frac{v_y}{v_x})}^{\tan^{-1}(\frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot X)} \sqrt{1+\tan^2(\theta)} \cdot -\frac{v_x^2}{g} \sec^2(\theta) d\theta$$

As $1 + \tan^2(\theta) \equiv \sec^2(\theta)$

$$\int_{\tan^{-1}(\frac{v_y}{v_x})}^{\tan^{-1}(\frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot X)} -\frac{v_x^2}{g} \sec^3(\theta) d\theta = -\frac{v_x^2}{g} \int_{\tan^{-1}(\frac{v_y}{v_x})}^{\tan^{-1}(\frac{v_y}{v_x} - \frac{g}{v_x^2} \cdot X)} \sec^2(\theta) \sec(\theta) d\theta$$

By parts:

$$u = \sec(\theta)$$
$$\frac{du}{dx} = \sec(\theta)\tan(\theta)$$
$$\frac{dv}{dx} = \sec^2(\theta)$$
$$v = \tan(\theta)$$
$$\int \sec^3(\theta)d\theta = \sec(\theta)\tan(\theta) - \int \sec(\theta)\tan^2(\theta)d\theta$$
$$\int \sec^3(\theta)d\theta = \sec(\theta)\tan(\theta) - \int \sec(\theta)(\sec^2(\theta) - 1)d\theta$$
$$\int \sec^3(\theta)d\theta = \sec(\theta)\tan(\theta) + \int \sec(\theta)d\theta - \int \sec^3(\theta)d\theta$$
$$2\int \sec^3(\theta)d\theta = \sec(\theta)\tan(\theta) + \ln(\sec(\theta) + \tan(\theta))$$
$$\int \sec^3(\theta)d\theta = \frac{\sec(\theta)\tan(\theta) + \ln(\sec(\theta) + \tan(\theta))}{2} + c$$

Substituting:

$$-\frac{v_x^2}{g}\left[\frac{\sec(\theta)\tan(\theta) + \ln(\sec(\theta) + \tan(\theta))}{2}\right]_{\tan^{-1}(\frac{v_y}{v_x})}^{\tan^{-1}(\frac{v_y}{v_x} - \frac{g}{v_x^2}\cdot X)}$$
$$\tan(\tan^{-1}(\theta)) = \theta$$
$$\sec(\tan^{-1}(\theta)) = \sqrt{\theta^2 + 1} \qquad D$$
$$-\frac{v_x^2}{g}\left[\frac{\theta\sqrt{\theta^2+1} + \ln\left(\sqrt{\theta^2+1} + \theta\right)}{2}\right]_{\frac{v_y}{v_x}}^{\frac{v_y}{v_x} - \frac{g}{v_x^2}\cdot X}$$

D Showing $\sec(\tan^{-1}(\theta)) = \sqrt{\theta^2+1}$

let $\tan^{-1}(\theta) = x$

$$\begin{aligned}
\theta &= \tan x \\
\theta &= \frac{\sin x}{\cos x} \\
\theta^2 &= \frac{\sin^2 x}{\cos^2 x} \\
\theta^2 + 1 &= \frac{\sin^2 x + \cos^2 x}{\cos^2 x} \\
\theta^2 + 1 &= \frac{1}{\cos^2 x} \\
\sqrt{\theta^2+1} &= \sec x
\end{aligned}$$

Recall $x = \tan^{-1}(\theta)$. Substituting:

$$\sqrt{\theta^2+1} = \sec(\tan^{-1}(\theta))$$

E Finding the turning points of the distance from the origin

$$D = \sqrt{(v_x \cdot t)^2 + (h + v_y \cdot t - \frac{1}{2} g \cdot t^2)^2}$$

$$D = \sqrt{v_x^2 \cdot t^2 + h^2 + v_y h \cdot t - \frac{1}{2} g h \cdot t^2 + v_y h \cdot t + v_y^2 \cdot t^2 - \frac{1}{2} g v_y t^3 - \frac{1}{2} g h \cdot t^2 - \frac{1}{2} g v_y t^3 + \frac{1}{4} g^2 t^4}$$

$$D = \sqrt{h^2 + 2 v_y h t + (v_x^2 + v_y^2 - g h) t^2 - g v_y t^3 + \frac{1}{4} g^2 t^4}$$

$$\frac{dD}{dt} = \frac{1}{2} \cdot \frac{2 v_y h + 2(v_x^2 + v_y^2 - g h) t - 3 g v_y t^2 + g^2 t^3}{\sqrt{h^2 + 2 v_y h t + (v_x^2 + v_y^2 - g h) t^2 - g v_y t^3 + \frac{1}{4} g^2 t^4}} = 0$$

$$2 v_y h + 2(v_x^2 + v_y^2 - g h) t - 3 g v_y t^2 + g^2 t^3 = 0$$

This can then be used in the cubic formula, where

$$a = g^2$$
$$b = -3gv_y$$
$$c = 2(v_x^2 + v_y^2 - gh)$$
$$d = 2v_y h$$
$$p = b^2 - 3ac$$
$$q = 2b^3 - 9abc + 27a^2 d$$
$$r = \sqrt[3]{\frac{q + \sqrt{q^2 - 4p^3}}{2}}$$
$$k_0 = 1$$
$$k_1 = \frac{-1 + \sqrt{-3}}{2}$$
$$k_2 = \frac{-1 - \sqrt{-3}}{2}$$
$$t = -\frac{1}{3a}\left(b + kr + \frac{p}{kr}\right)$$

Each value of k gives a unique solution. We can only plot the ones over the real range we are plotting. However, evidently these equations involve complex numbers, not natively supported in JavaScript. I therefore created my own complex number class, discussed here F.

F Creating a complex number class

I decided to implement complex numbers using two real numbers, a and b, making up the complex number in the form $a + bi$. Alternatives do exist, like polar form, which may have some trade-offs in terms of efficiency, but the efficiency of the calculations aren't a large concern to me as computations involving complex numbers are only done to find the turning points of the graph and aren't used to calculate every single point on the graph. (Unlike some of the other algorithms I have implemented which need to be more efficiency-aware)

Implementing functions like adding and multiplying complex numbers were relatively trivial. However, one of the functions I needed was for dividing a real number by a complex number. How I worked out this function is shown here, where c and d are the

outputs of the function:

$$\frac{k}{a+bi} = c + di$$
$$k = ac - bd + adi + bci$$

Comparing coefficients:

$$k = ac - bd$$
$$c = \frac{k + bd}{a}$$
$$0 = ad + bc$$
$$0 = ad + b(\frac{k + bd}{a})$$
$$0 = a^2d + bk + b^2d$$
$$d = -\frac{bk}{a^2 + b^2}$$

This can then be substituted to attain the value for c.

Another problem I encountered was calculating the three complex roots of a real number. It is convenient to solve this problem in polar form, and then convert it back to rectangular form. Say we want to find the cube root of x. In polar form, all the following forms are equivalent:

$$re^{i\theta}$$
$$re^{i(\theta+2\pi)}$$
$$re^{i(\theta-2\pi)}$$

This is because rotating by two pi radians brings you back where you started. However, when we take their cube roots, we attain three different results:

$$\sqrt[3]{r}e^{\frac{i\theta}{3}}$$
$$\sqrt[3]{r}e^{\frac{i(\theta+2\pi)}{3}}$$
$$\sqrt[3]{r}e^{\frac{i(\theta-2\pi)}{3}}$$

These are the three results we are looking for. I therefore implemented an algorithm to transform rectangular to polar form, do the transformations, and then return it to rectangular form.

G Finding the first intersection point between a line and a sphere

Substituting each entry of the line equation into the sphere equation yields:

$$x = (p_0 - d)t + d$$
$$y = p_1 t$$
$$z = p_2 t$$
$$x^2 + y^2 + z^2 = r^2$$
$$((p_0 - d)t + d)^2 + (p_1 t)^2 + (p_2 t)^2 = r^2$$
$$(p_0 - d)^2 t^2 + 2d(p_0 - d)t + d^2 + p_1^2 t^2 + p_2^2 t^2 = r^2$$
$$((p_0 - d)^2 + p_1^2 + p_2^2)t^2 + 2d(p_0 - d)t + d^2 - r^2 = 0$$
$$t = \frac{-2d(p_0 - d) \pm \sqrt{(2d(p_0 - d))^2 - 4((p_0 - d)^2 + p_1^2 + p_2^2)(d^2 - r^2)}}{2((p_0 - d)^2 + p_1^2 + p_2^2)}$$
$$t = \frac{-d(p_0 - d) \pm \sqrt{d^2(p_0 - d)^2 - ((p_0 - d)^2 + p_1^2 + p_2^2)(d^2 - r^2)}}{((p_0 - d)^2 + p_1^2 + p_2^2)}$$

We are looking for the smaller value of t (the intersection that is closer to the camera). The denominator is positive, as it is the sum of squares. As the square root will always be positive, if we subtract the square root from something, the result will always be smaller than if we add the square root to something. Therefore, the smaller value of t occurs when we use the negative sign.[15]

$$t = \frac{-d(p_0 - d) - \sqrt{d^2(p_0 - d)^2 - ((p_0 - d)^2 + p_1^2 + p_2^2)(d^2 - r^2)}}{((p_0 - d)^2 + p_1^2 + p_2^2)}$$

x, y and z can then be attained using t from the first three lines of this working.

H Basic linear algebra

H.1 Matrix-vector multiplication

A matrix can be considered to be a way to transform a vector, where each entry in the new vector is dependent on a weighted sum of each of the previous values. For example, a matrix may describe some transformation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \mapsto \begin{bmatrix} 2x + 3y - z \\ 1x + 0y - 10z \\ -3x + 5y + z \end{bmatrix}$$

[15] If one of the two ts are negative, this indicates that the camera is inside the earth. In this scenario, no output would be meaningful, so we can assume that both values of t are positive.

In order to represent this transformation we write out each coefficient in a table as shown:

$$\begin{bmatrix} 2 & 3 & -1 \\ 1 & 0 & -10 \\ -3 & 5 & 1 \end{bmatrix}$$

And to represent some vector being transformed by the matrix, the matrix is written before the vector, so:

$$\begin{bmatrix} 2 & 3 & -1 \\ 1 & 0 & -10 \\ -3 & 5 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2x + 3y - z \\ 1x + 0y - 10z \\ -3x + 5y + z \end{bmatrix}$$

H.2 The identity matrix

The identity matrix is a matrix with a special property that, when multiplied by any vector, the result is unchanged. This is a matrix in which every value along the diagonal length from top left to bottom right is a 1 and every other value is 0, as shown:

$$\begin{bmatrix} 1 & 0 & 0\dots \\ 0 & 1 & 0\dots \\ 0 & 0 & 1\dots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{1}$$

Every entry gets multiplied by one and placed into the new entry row while all the others entries get multiplied by zero and so have no impact, visible here:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \cdot x + 0 \cdot y + 0 \cdot z \\ 0 \cdot x + 1 \cdot y + 0 \cdot z \\ 0 \cdot x + 0 \cdot y + 1 \cdot z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

H.3 Matrix-matrix multiplication

If you were to take two successive matrix transformations of some vector, it would be helpful to condense this whole transformation into a single matrix. This is done by, for each entry in the resultant matrix, summing the products of the corresponding elements from the two original matrices, one taken in rows, the other taken in columns. For example:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} = \begin{bmatrix} ai + bk & aj + bl \\ ci + dk & cj + dl \end{bmatrix}$$

I General rotation matrix

The general rotation matrix rotates a point (x, y, z) about an axis defined by the unitary vector (u_x, u_y, u_z) by angle θ through matrix-vector multiplication. It is given here:

$$\begin{bmatrix} u_x^2\,(1-\cos\theta)+\cos\theta & u_x u_y\,(1-\cos\theta)-u_z\sin\theta & u_x u_z\,(1-\cos\theta)+u_y\sin\theta \\ u_x u_y\,(1-\cos\theta)+u_z\sin\theta & u_y^2\,(1-\cos\theta)+\cos\theta & u_y u_z\,(1-\cos\theta)-u_x\sin\theta \\ u_x u_z\,(1-\cos\theta)-u_y\sin\theta & u_y u_z\,(1-\cos\theta)+u_x\sin\theta & u_z^2\,(1-\cos\theta)+\cos\theta \end{bmatrix}$$