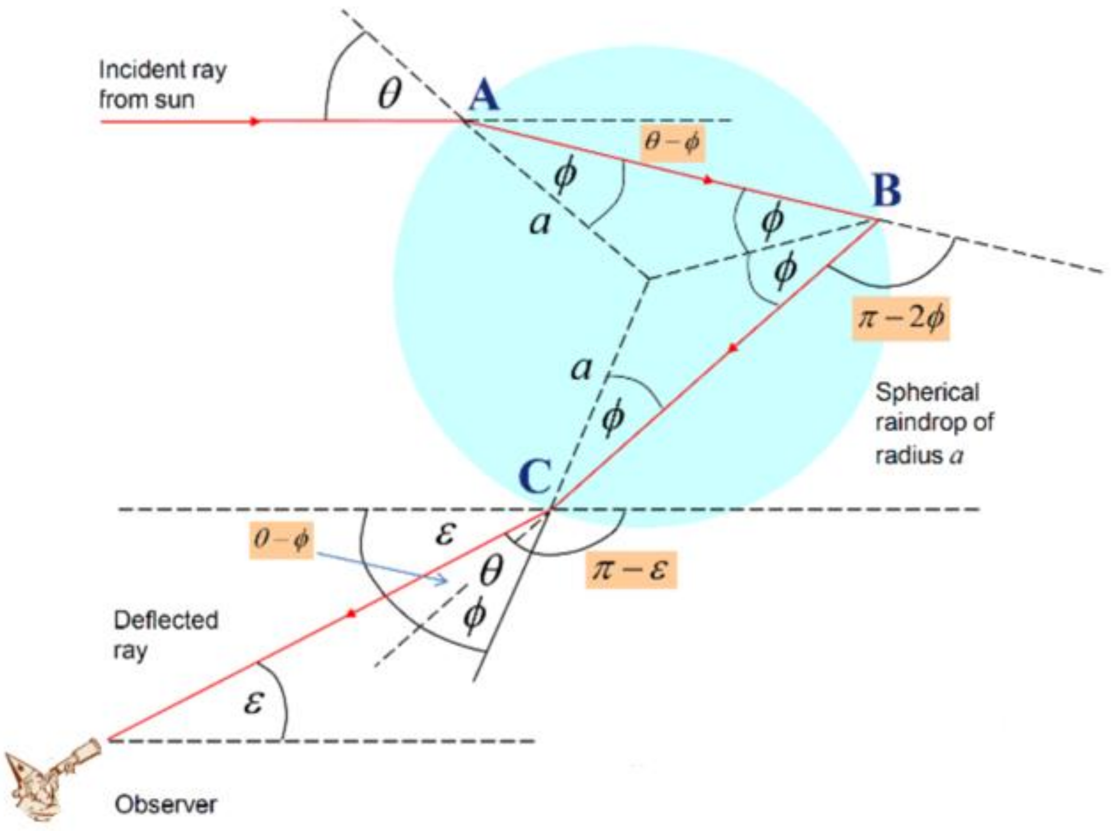
Rainbows Physics: modelled using computers

Rainbows have long amazed humanity. First they were believed to be bridges to other realms, then they were religious symbols of divine promises, and now they hold important cultural (and political) significance, both in modern Christianity and notably the LGBTQ movement, despite having been de-mystified by science. The first man to properly define them mathematically, René Descartes.

In his paper Meteorologica, Aristotle wanted to know four keys things about rainbows: what is involved in a rainbow's formation, what determined its shape, its size, and its colours. He identified the 'agents of formation' as the sun, a rain shower, and an observer. He also said that the shape was a circular arc. These were the ideas of his that weren't misleading, all the others were. He was the first recorded to investigate the phenomenon, but Descartes was the man who made a very large leap in understanding where they come from.

He knew that internal reflection inside a raindrop was probably responsible, and so we consider this simplified model of a raindrop as a 2D circle:



(In this diagram, taken from the BPhO 2025 Challenge document, only the relevant rays are shown in red.)

On incidence with the water droplet (at A), some of the light may reflect while the rest refracts into the droplet, at an angle φ which depends on the refractive index of the water *and on the frequency of the light* (https://opticalcalculator.com/task-1.html). On reaching B, the relevant light internally reflects back into the droplet at the same angle φ, as the law of reflection states. Finally, on reaching C, the drop refracts back out of the droplet and into the eye of the observer, who sees a rainbow. The reason for the spread of colours is that, on refraction, each frequency of light refracts at a very slightly different angle, owing to the fact that the refractive indices of materials change for different frequencies of light. The double refraction seen here is similar to that which occurs in a prism (demonstrated here: https://opticalcalculator.com/task-12.html) in which the different colours which comprise white light separate into the familiar spectrum of colours.

To find the angle ε at which the deflected ray meets the observer, all one has to do is to do some easy angle addition: The ray turns three times, at A it turns clockwise through an angle of θ – φ, at B it turns clockwise by π – 2φ, and at C it turns clockwise again by θ – φ (as illustrated). Therefore its angle of deviation from the horizontal (where it started) is θ – φ + π + 2φ + θ – φ (which is just the sum of all the turns, as they are all clockwise.). This angle of deviated is equal to π – ε, and so we end up with ε = 4φ – 2θ. We can graph what this leads to, namely that different frequencies lead to slightly different ε in both primary and secondary rainbows.

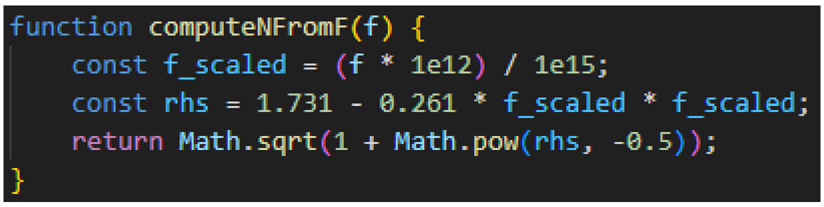
A quick note on secondary rainbows: a secondary rainbow is formed when light rays reflect twice inside the raindrop, which reverses the order of colours seen and leads to the secondary being above the primary in the observer's eye. The formula for $ε_2$ is π – 6φ + 2θ. This expression is found in the same way as the primary, only with two internal reflections instead of one.

First, we need to find the refractive index for a given inputted frequency, this is done using the empirical formula:

$$\left(n^2 - 1\right)^{-2} = 1.731 - 0.261\left(\frac{f}{10^{15}\mathrm{Hz}}\right)^2$$

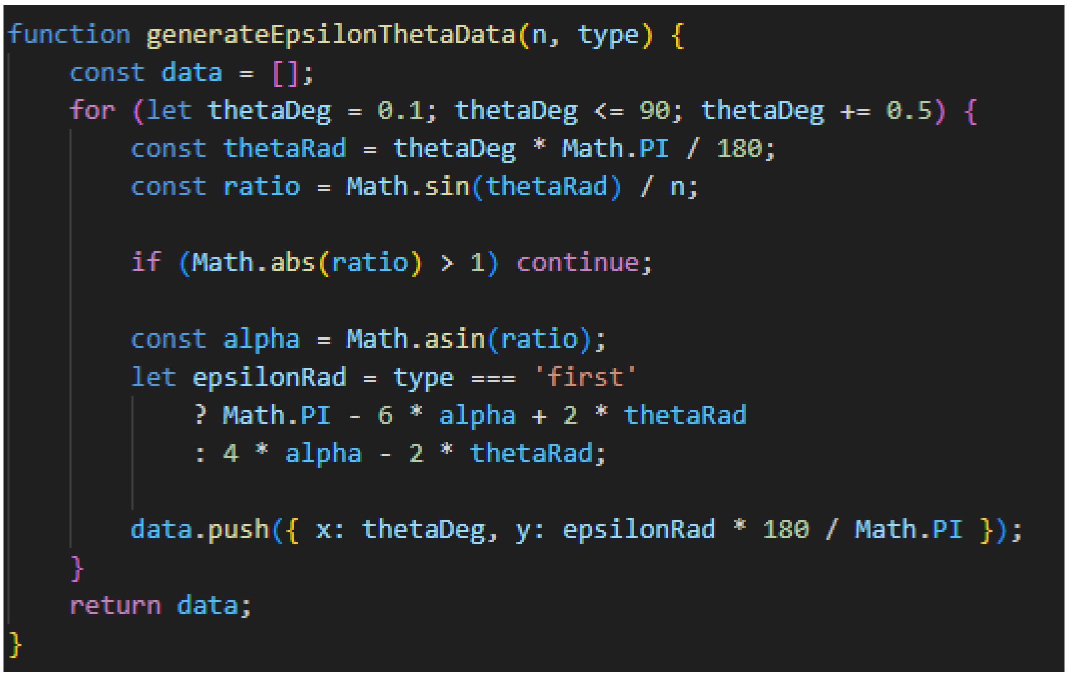
This formula is used to graph refractive index vs frequency at https://opticalcalculator.com/task-1.html (task b).

We incorporate this equation into a function called computeNfromF(), which takes a frequency parameter and returns refractive index:



```
function computeNFromF(f) {
    const f_scaled = (f * 1e12) / 1e15;
    const rhs = 1.731 - 0.261 * f_scaled * f_scaled;
    return Math.sqrt(1 + Math.pow(rhs, -0.5));
}
```

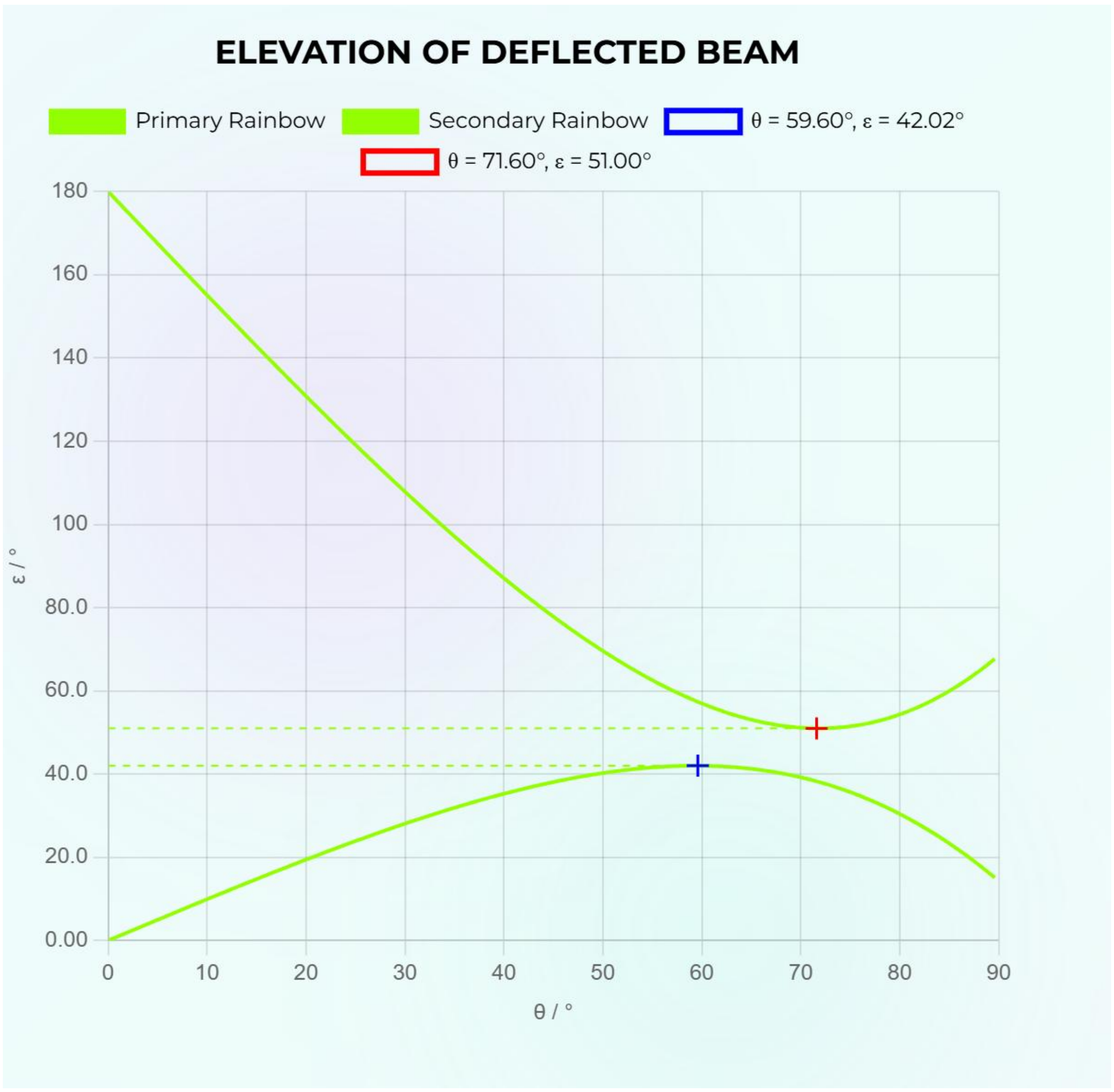
Using the result of this function, we can calculate values of φ for each value of θ from 0 to 90 degrees, using Snell's law ($n_1 \sin\theta_1 = n_2 \sin\theta_2$). We can also therefore calculate ε for given angles of θ, for both types of rainbow. This is all incorporated into the following function:



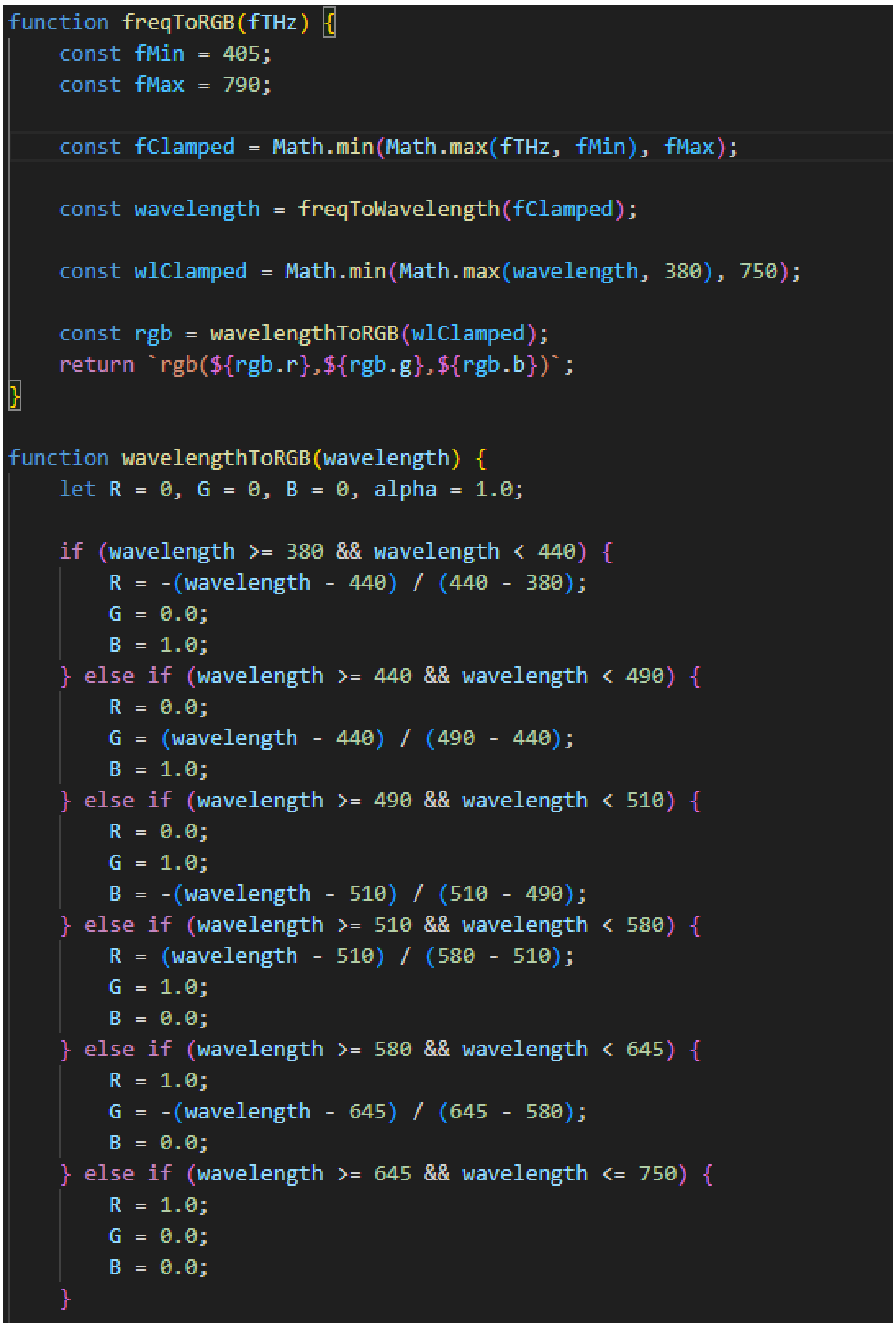
```
function generateEpsilonThetaData(n, type) {
    const data = [];
    for (let thetaDeg = 0.1; thetaDeg <= 90; thetaDeg += 0.5) {
        const thetaRad = thetaDeg * Math.PI / 180;
        const ratio = Math.sin(thetaRad) / n;

        if (Math.abs(ratio) > 1) continue;

        const alpha = Math.asin(ratio);
        let epsilonRad = type === 'first'
            ? Math.PI - 6 * alpha + 2 * thetaRad
            : 4 * alpha - 2 * thetaRad;

        data.push({ x: thetaDeg, y: epsilonRad * 180 / Math.PI });
    }
    return data;
}
```

The 'type' variable manages whether the rainbow being calculated for is secondary or primary, and does the necessary calculation. Θ is converted to radians, and then φ is calculated using Snell's law (here called alpha for simplicity). The values of ε are added to the 'data' array and this is then returned, to be graphed using chart.js.

With chart.js configured and a plugin used to find the stationary points of both curves and mark them with a red dashed line and cross, we get the following graph:



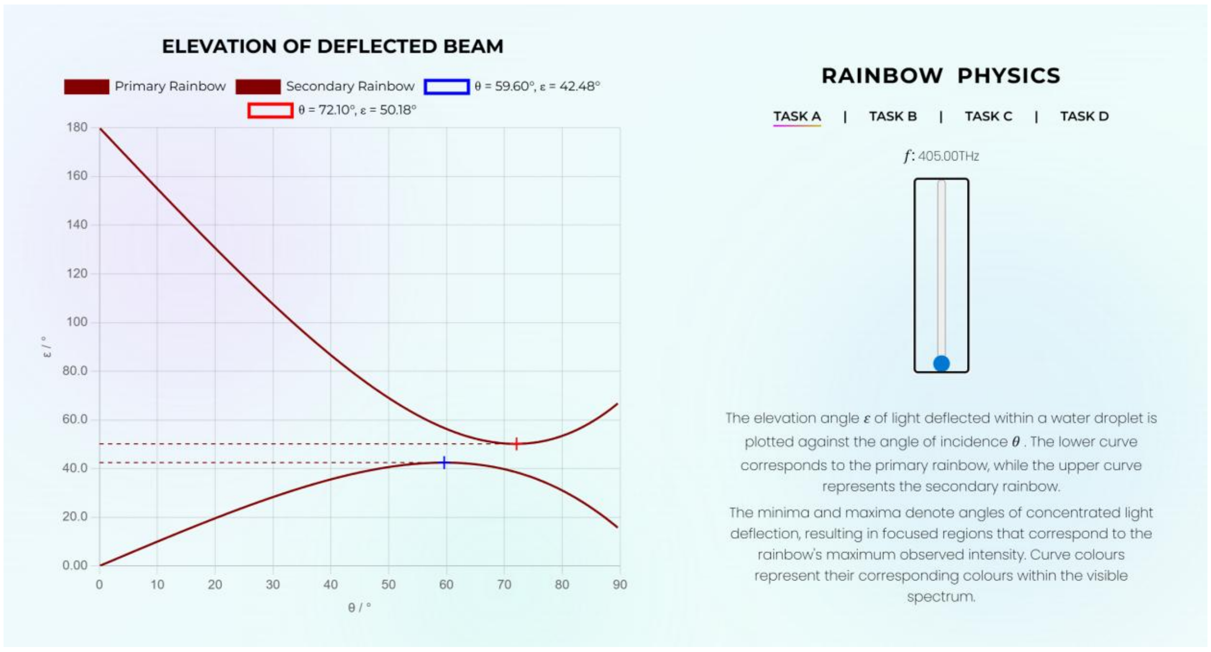
The colour coding was added using the following functions, which return an rgb value for the chart.js module to use for the plotted line, for a given frequency or wavelength input:



```
function freqToRGB(fTHz) {
    const fMin = 405;
    const fMax = 790;

    const fClamped = Math.min(Math.max(fTHz, fMin), fMax);

    const wavelength = freqToWavelength(fClamped);

    const wlClamped = Math.min(Math.max(wavelength, 380), 750);

    const rgb = wavelengthToRGB(wlClamped);
    return `rgb(${rgb.r},${rgb.g},${rgb.b})`;
}

function wavelengthToRGB(wavelength) {
    let R = 0, G = 0, B = 0, alpha = 1.0;

    if (wavelength >= 380 && wavelength < 440) {
        R = -(wavelength - 440) / (440 - 380);
        G = 0.0;
        B = 1.0;
    } else if (wavelength >= 440 && wavelength < 490) {
        R = 0.0;
        G = (wavelength - 440) / (490 - 440);
        B = 1.0;
    } else if (wavelength >= 490 && wavelength < 510) {
        R = 0.0;
        G = 1.0;
        B = -(wavelength - 510) / (510 - 490);
    } else if (wavelength >= 510 && wavelength < 580) {
        R = (wavelength - 510) / (580 - 510);
        G = 1.0;
        B = 0.0;
    } else if (wavelength >= 580 && wavelength < 645) {
        R = 1.0;
        G = -(wavelength - 645) / (645 - 580);
        B = 0.0;
    } else if (wavelength >= 645 && wavelength <= 750) {
        R = 1.0;
        G = 0.0;
        B = 0.0;
    }
```

With a frequency slider added, one can see that as the frequency is increased, ε_1 decreases and ε_2 increases:

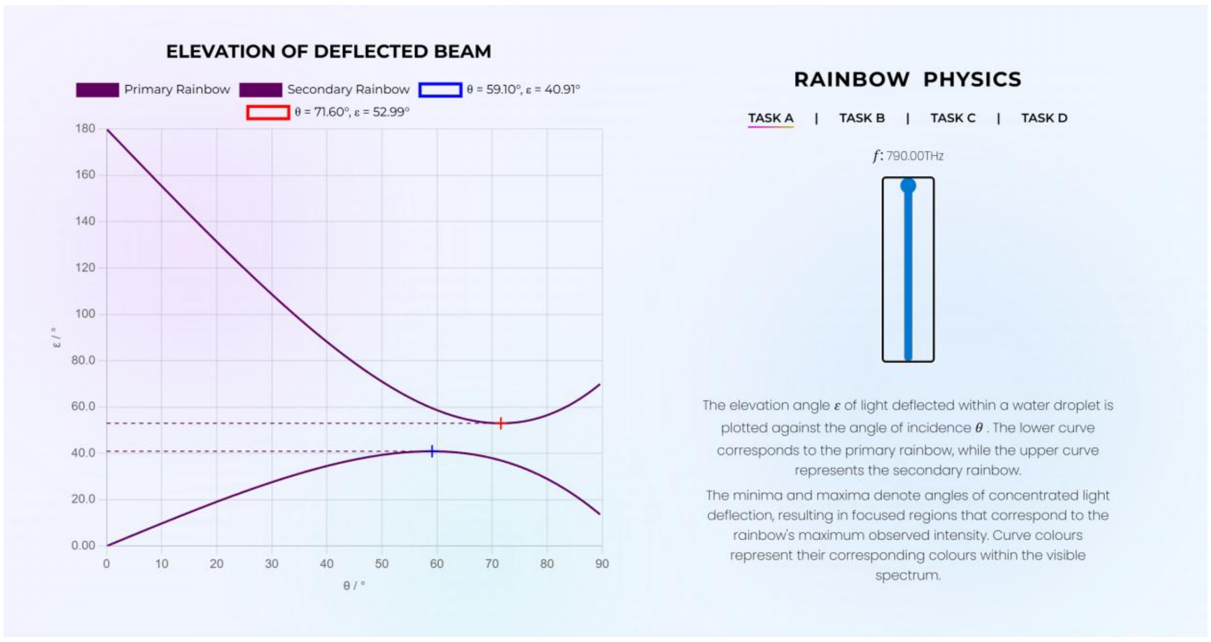
Minimum f:



The elevation angle *ε* of light deflected within a water droplet is plotted against the angle of incidence *θ*. The lower curve corresponds to the primary rainbow, while the upper curve represents the secondary rainbow.

The minima and maxima denote angles of concentrated light deflection, resulting in focused regions that correspond to the rainbow's maximum observed intensity. Curve colours represent their corresponding colours within the visible spectrum.

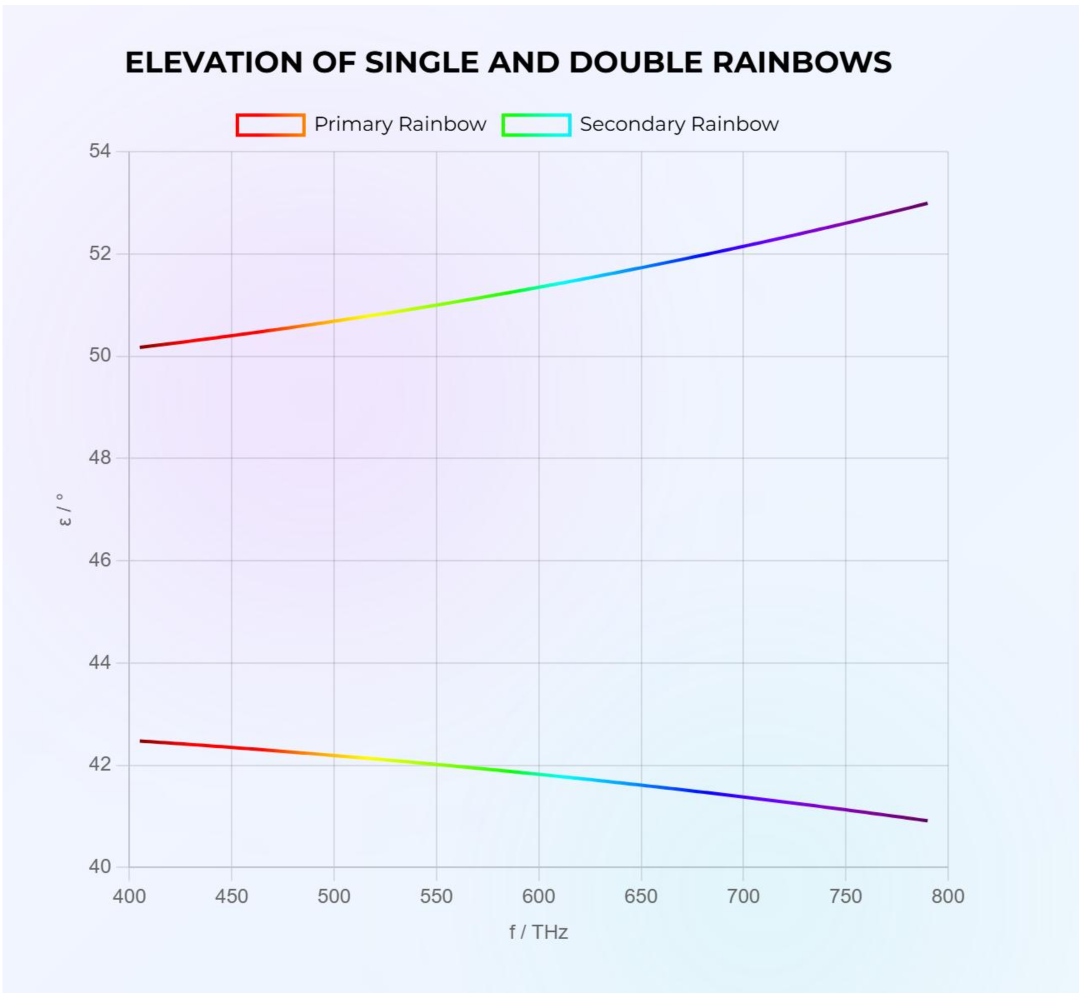
Maximum f:



The elevation angle *ε* of light deflected within a water droplet is plotted against the angle of incidence *θ*. The lower curve corresponds to the primary rainbow, while the upper curve represents the secondary rainbow.

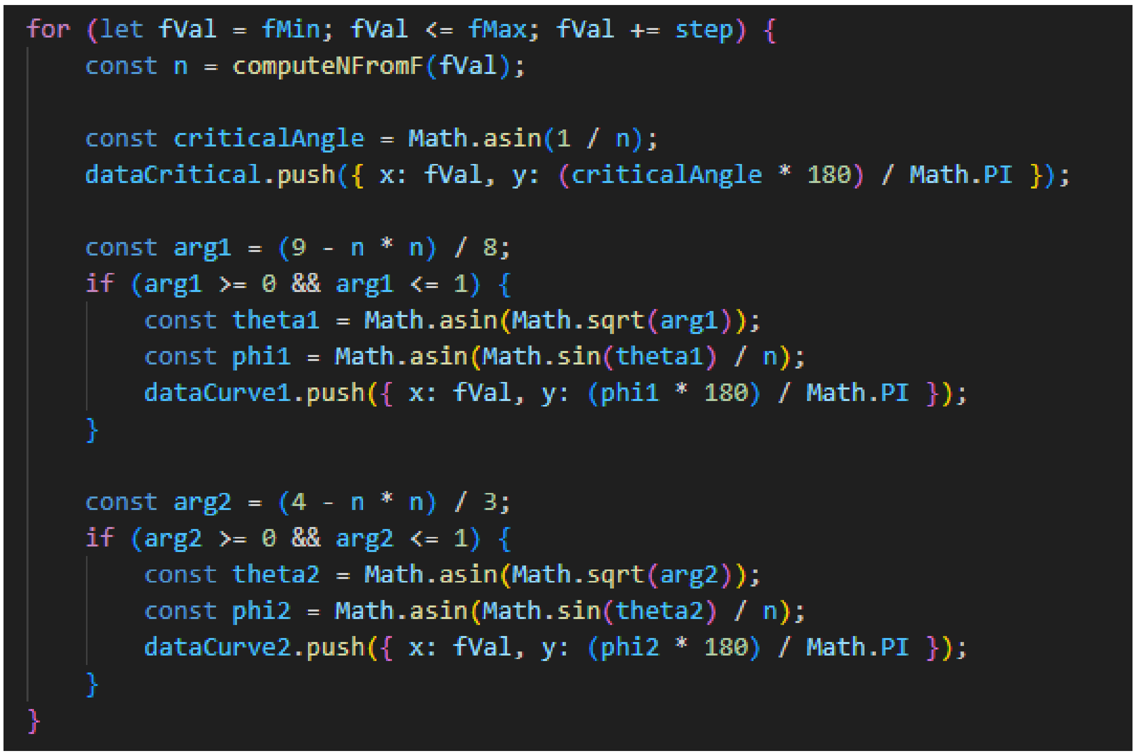
The minima and maxima denote angles of concentrated light deflection, resulting in focused regions that correspond to the rainbow's maximum observed intensity. Curve colours represent their corresponding colours within the visible spectrum.

We can also plot ε vs frequency, to see how each colour of light is positioned differently in the rainbow and show that the colour order of primary and secondary rainbows is reversed. All we do is replace the θ in the x axis array with frequency in terahertz, and set a constant angle of incidence, then run similar functions to find our values in the range of frequencies in which light is visible:



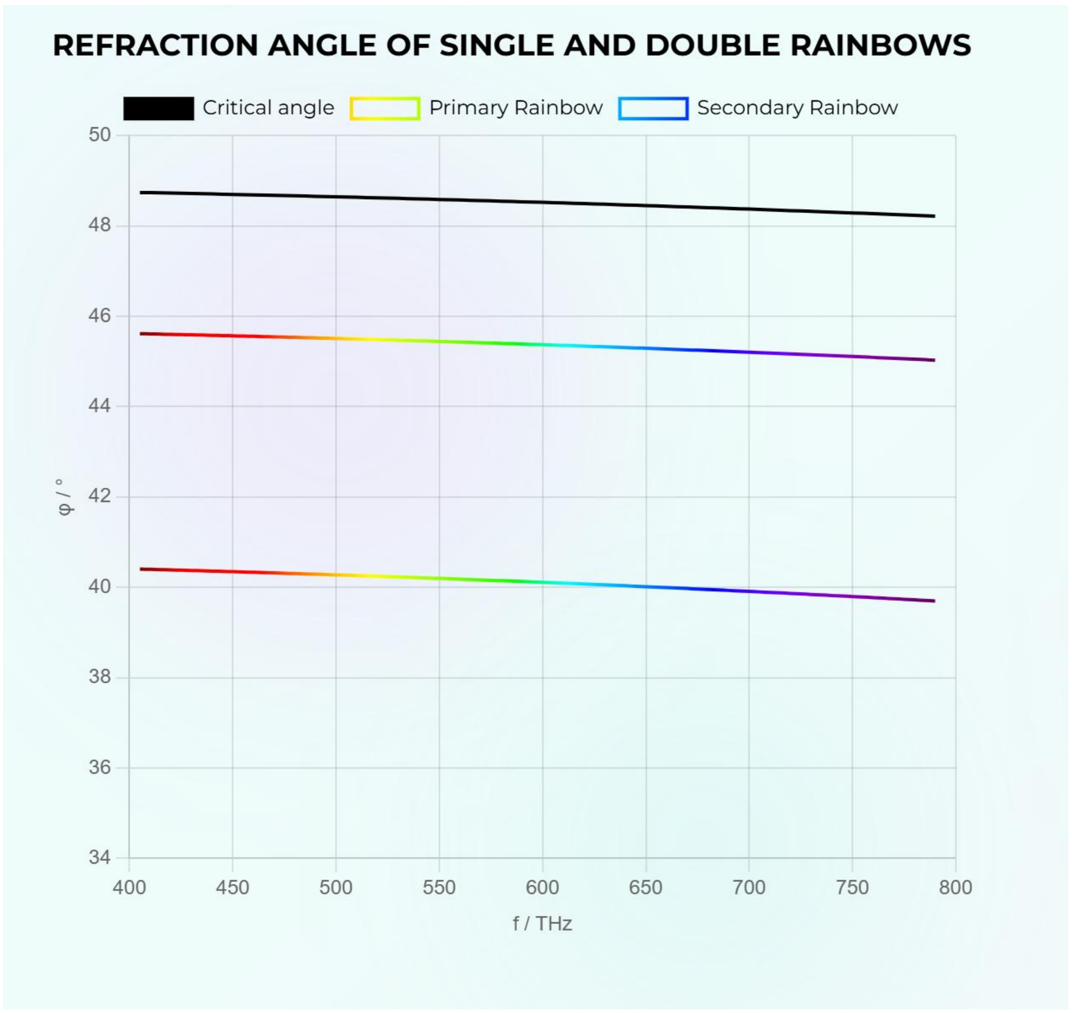
Colour coding again was using the freqtoRGB and wavelengthtoRGB functions.

Secondary rainbows are fainter to the eye than their respective primaries, and we are able to demonstrate why this is. If we calculate the angle of refraction on the way out of the droplet for primary and secondary rainbow rays, and then also plot the critical angle, using the following function…



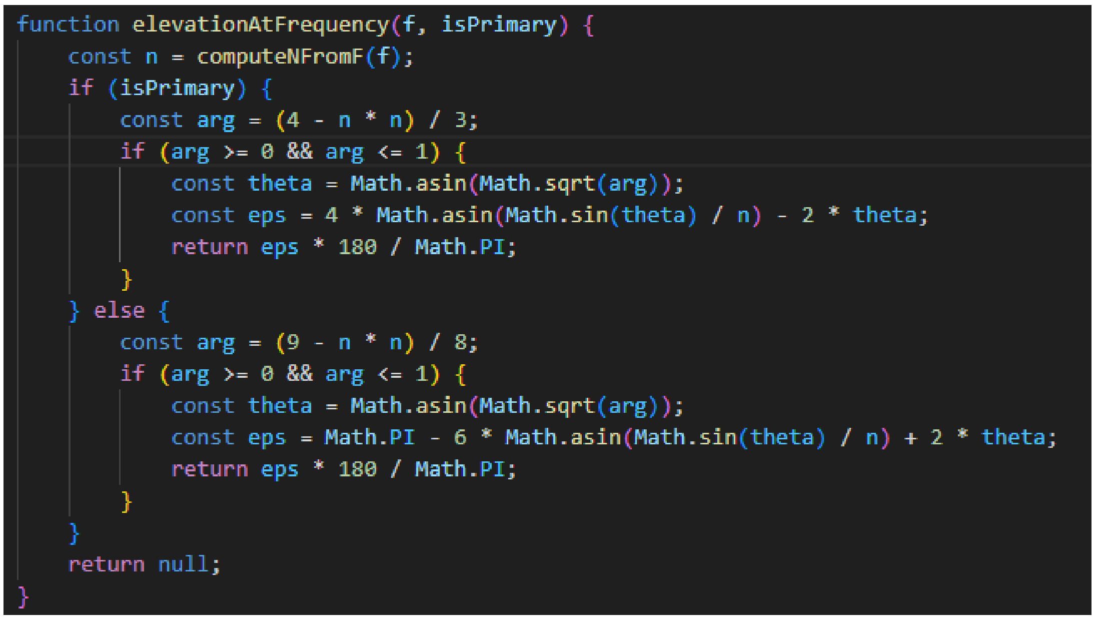
```
for (let fVal = fMin; fVal <= fMax; fVal += step) {
    const n = computeNFromF(fVal);

    const criticalAngle = Math.asin(1 / n);
    dataCritical.push({ x: fVal, y: (criticalAngle * 180) / Math.PI });

    const arg1 = (9 - n * n) / 8;
    if (arg1 >= 0 && arg1 <= 1) {
        const theta1 = Math.asin(Math.sqrt(arg1));
        const phi1 = Math.asin(Math.sin(theta1) / n);
        dataCurve1.push({ x: fVal, y: (phi1 * 180) / Math.PI });
    }

    const arg2 = (4 - n * n) / 3;
    if (arg2 >= 0 && arg2 <= 1) {
        const theta2 = Math.asin(Math.sqrt(arg2));
        const phi2 = Math.asin(Math.sin(theta2) / n);
        dataCurve2.push({ x: fVal, y: (phi2 * 180) / Math.PI });
    }
}
```

… and plot and colour code the data arrays, the following chart arises:



In which the primary rainbow is the higher line on the graph. The fact that it is closer to the critical angle is interesting, because as mentioned above, not all the light internally reflects unless the critical angle is lesser than the angle of incidence. As this is not the case for either, both the primary and secondary rainbow are not at full intensity. More interestingly, because the secondary angle is much smaller than the primary, more of it refracts each time it reflects within the droplet, leading to more loss in intensity and hence a weaker rainbow to an observer.

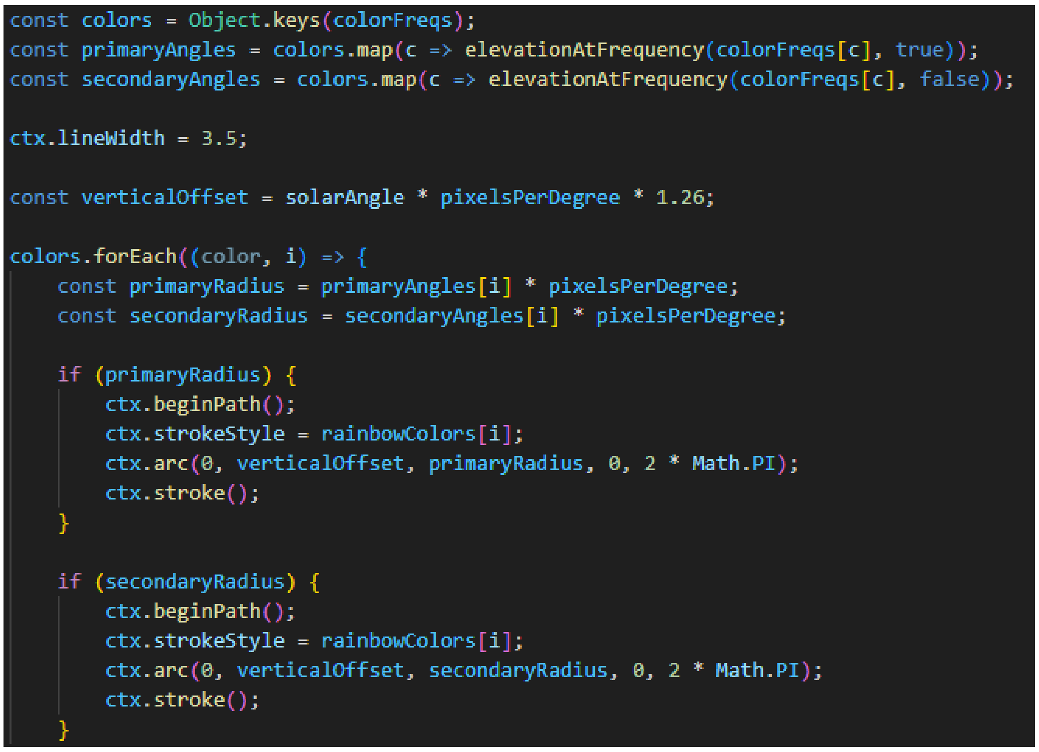
Finally, we can plot the actual position of a rainbow relative to the horizon, for a given solar angle α. We calculate ε using previous code (calculating φ then using addition to find ε), and add this to a function called elevationatfrequency:



```
function elevationAtFrequency(f, isPrimary) {
    const n = computeNFromF(f);
    if (isPrimary) {
        const arg = (4 - n * n) / 3;
        if (arg >= 0 && arg <= 1) {
            const theta = Math.asin(Math.sqrt(arg));
            const eps = 4 * Math.asin(Math.sin(theta) / n) - 2 * theta;
            return eps * 180 / Math.PI;
        }
    } else {
        const arg = (9 - n * n) / 8;
        if (arg >= 0 && arg <= 1) {
            const theta = Math.asin(Math.sqrt(arg));
            const eps = Math.PI - 6 * Math.asin(Math.sin(theta) / n) + 2 * theta;
            return eps * 180 / Math.PI;
        }
    }
    return null;
}
```

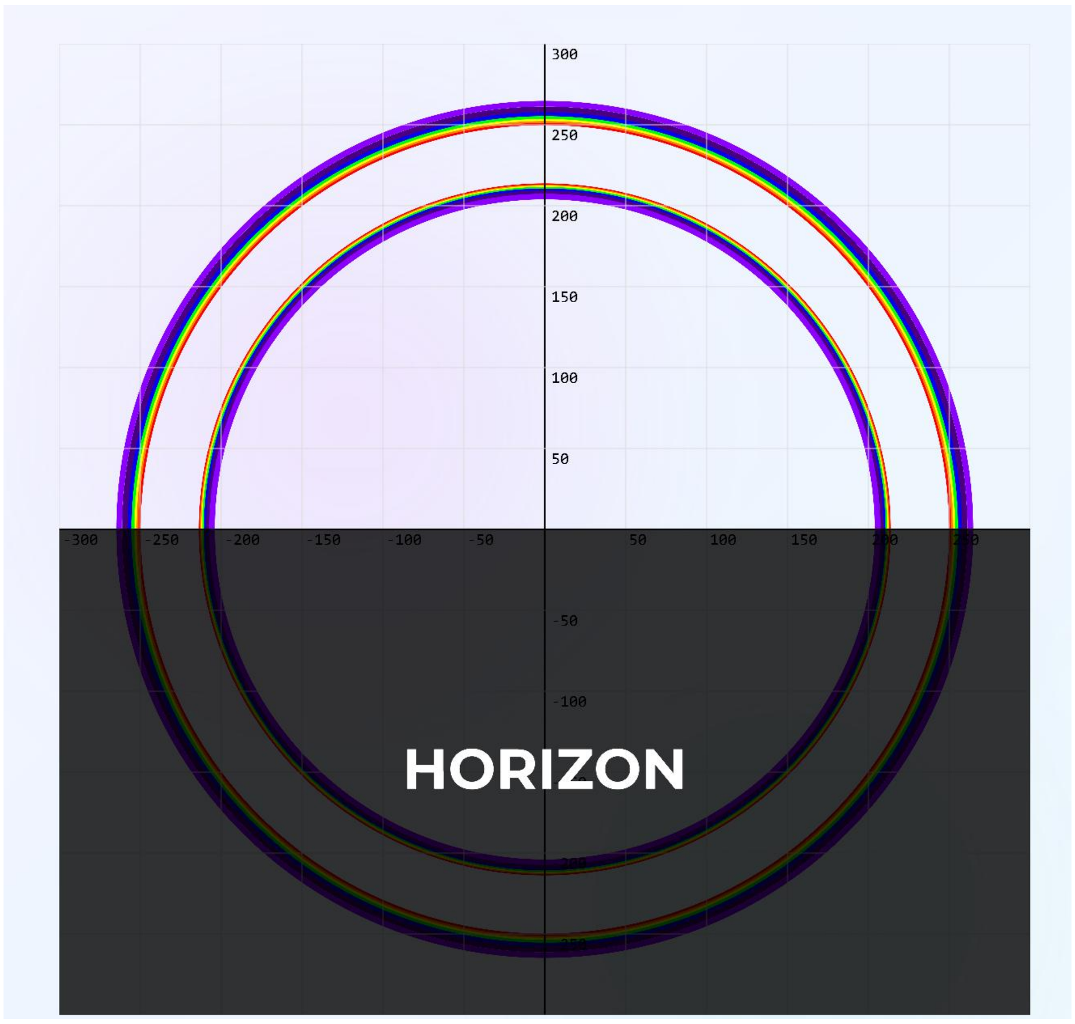
which calculates ε for a given frequency (and inputted value of α).

Then, we create a list of colours, and for each colour draw a circle of radius primary and radius secondary, having calculated the radii using a conversion of pixels per degree, and offset it by the correct amount:

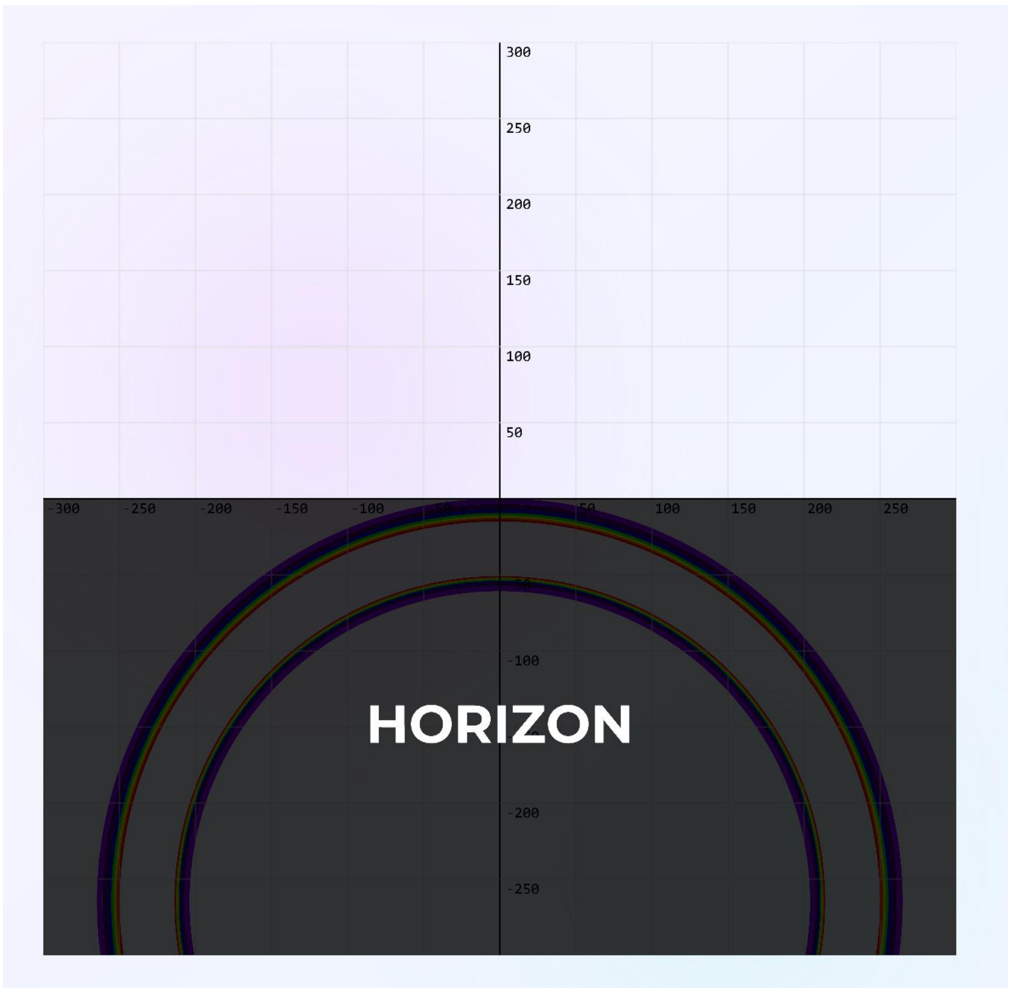


```
const colors = Object.keys(colorFreqs);
const primaryAngles = colors.map(c => elevationAtFrequency(colorFreqs[c], true));
const secondaryAngles = colors.map(c => elevationAtFrequency(colorFreqs[c], false));

ctx.lineWidth = 3.5;

const verticalOffset = solarAngle * pixelsPerDegree * 1.26;

colors.forEach((color, i) => {
    const primaryRadius = primaryAngles[i] * pixelsPerDegree;
    const secondaryRadius = secondaryAngles[i] * pixelsPerDegree;

    if (primaryRadius) {
        ctx.beginPath();
        ctx.strokeStyle = rainbowColors[i];
        ctx.arc(0, verticalOffset, primaryRadius, 0, 2 * Math.PI);
        ctx.stroke();
    }

    if (secondaryRadius) {
        ctx.beginPath();
        ctx.strokeStyle = rainbowColors[i];
        ctx.arc(0, verticalOffset, secondaryRadius, 0, 2 * Math.PI);
        ctx.stroke();
    }
```

With the ‘horizon’ shaded at y=0 for illustrative purposes, as that is what a rainbow is, a circle partially obscured by the horizon. We end up with the following:



At minimum solar angle (ie sunset/rise), and at the maximum solar angle we get this :



Noting that this maximum solar angle is 42 degrees, when a rainbow is the strongest looking to an observer. Try it for yourself here: https://opticalcalculator.com/task-11.html (task D).

All coding and graphic illustrations taken from opticalcalculator.com, all scientific diagrammatic material taken from Bpho Computational challenge: 2025 Geometric Optics (Attached below).

BPhO ComPhys Challenge 2025.pdf