Development of Code for Simulating Vavilov-Cherenkov Radiation

Aleksas Girenas

March 2021

1 Abstract

In this paper we discuss the importance of Vavilov-Cherenkov Radiation and develop code in **C++** and **Python** to simulate polarisation radiation using the generalised equation developed by M. V. Shevelev and A. S. Konkov [1]. Using the code we demonstrate that polarisation radiation increases quadratically with increasing size 'a' of the prismatic target and demonstrate the effectiveness. Finally we simulate polarisation radiation for Diamond Light Source proving the simulations use and importance for assisting in particle and material analysis particularly in the development of beam position monitors. The full project source code can be found on Github.

2 Introduction

Cherenkov radiation refers to the emission of electromagnetic radiation from a charged particle traveling inside a dielectric at a speed greater than the speed of light in the medium. It is a phenomena analogous to sound waves being supersonic in a medium and was first observed experimentally by Cherenkov in 1934 and later developed theoretically by Tamm and Frank in 1937, confirming these initial observations [2]. The theory of Cherenkov radiation was further developed in 1947 by Frank and Ginzburg [3] who considered the emission of Cherenkov radiation from particles moving parallel to a dielectric target with permittivity ϵ. In this paper, we refer to polarisation radiation as the radiation emitted through the Cherenkov effect by particles traveling at a distance b from the surface of a dielectric, also called the impact parameter.

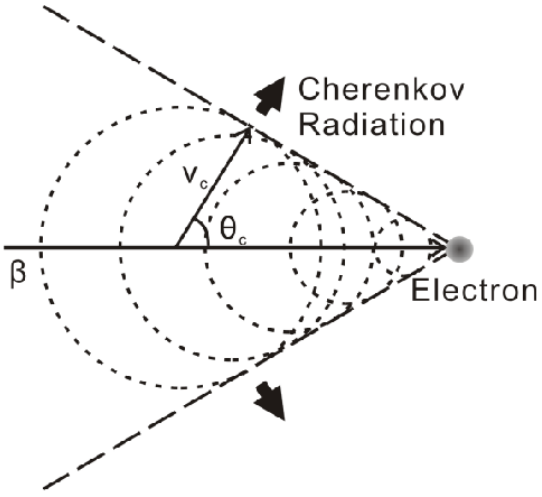

Figure 1: Cherenkov radiation produced by a relativistic electron moving through a dielectric medium at a constant speed [4]

Cherenkov radiation from relativistic charged particles has led to the development of applications in many fields including astrophysics and particle detection and identification [5]. The light intensity scales proportionally to the length of the radiator and so we can produce a large photon flux when using a long radiator [6]. The light is emitted at a characteristic angle θ, defined as $n\beta cos\Theta = 1$ depending on particle velocity β and the index of refraction of the radiator n. This angle is usually large and so limits the contamination of the Cherenkov signal by other sources of background light as we can steer the photons away from the particle beam trajectory. With such characteristics, Cherenkov radiation has also been regularly used in the instrumentation for charged particle accelerators; it is particularly important in beam position monitors that are non-destructive diagnostics used most frequently at nearly all linacs, cyclotrons, and synchrotrons [7].

There have, however, been very few exact solutions to the Vavilov-Cherenkov Radiation problem for media with sharp boundaries which is important for use in synchrotrons. The generalised generation of Diffraction Radiation (DR) and Vavilov-Cherenkov Radiation (VCR) has recently been addressed in the equation developed by M. V. Shevelev and A. S. Konkov [1] which we will use throughout this paper. We will apply the solution to VCR occurring from the uniform motion of a point charge in vacuum in the vicinity of a finite sized prismatic target with an arbitrary permittivity ϵ and analyse the distribution.

3 Vavilov-Cherenkov Radiation

In figure 2 we see the setup for producing polarisation radiation when a charged particle moves at a constant velocity near a prismatic wedge.

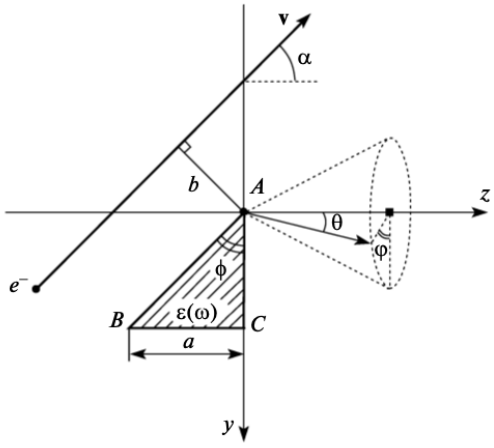

Figure 2: Generation of polarization radiation by a charged particle moving uniformly near a prismatic wedge [1]

The generalised equation [1] for the spectral angular distribution of polarised radiation for figure 2 in the positive direction of the z axis has the form:

$$\frac{d^2W}{d\omega d\Omega} = \frac{e^2\beta^2}{4\pi^2 c}\frac{\cos^2(\theta'-\alpha)}{|P|^2}\left|\frac{\varepsilon-1}{\varepsilon}\right|^2\left|1-\frac{P\exp\left[i\frac{\omega}{\beta c}\Sigma a\cot\phi\right]+\Sigma\cot\phi\exp\left[-ia\frac{\omega}{\beta c}P\right]}{P+\Sigma\cot\phi}\right|^2\Bigg] - PartA$$

$$\times\Bigg\{\left|\frac{\varepsilon}{\varepsilon\cos(\theta'-\alpha)+\sqrt{\varepsilon-\sin^2(\theta'-\alpha)}}\right|^2\Big|\cos\alpha\left(\gamma^{-1}\sin(\theta'-\alpha)-iK\cos\varphi\sqrt{\varepsilon-\sin^2(\theta'-\alpha)}\right)$$

$$+\sin\alpha\left(iK\sin(\theta'-\alpha)+\gamma^{-1}\cos\varphi\sqrt{\varepsilon-\sin^2(\theta'-\alpha)}\right)-\gamma\beta\sin(\theta'-\alpha)\sqrt{\varepsilon-\sin^2(\theta'-\alpha)}\sin^2\varphi\Big|^2$$

$$+\left|\frac{\sqrt{\varepsilon}}{\cos(\theta'-\alpha)+\sqrt{\varepsilon-\sin^2(\theta'-\alpha)}}\right|^2(\gamma\sin\varphi)^2\left(\sin^2(\theta'-\alpha)+\left|\sqrt{\varepsilon-\sin^2(\theta'-\alpha)}\right|^2\right)$$

$$\times\left[1-\beta^2\cos^2(\theta'-\alpha)+2\beta\gamma^{-2}\sin\alpha\sin(\theta'-\alpha)\cos\varphi-\gamma^{-2}\sin^2\alpha\left(K^2-\gamma^{-2}\right)\right]\Bigg\}\Bigg] - PartB$$

$$\exp\left[-2\tfrac{\omega}{\gamma\beta c}(h+a\cot\phi)K\cos\alpha\right]$$

$$K^2\left(1-\beta^2\cos^2(\theta'-\alpha)+\beta^2\sin^2\alpha\left[1-\sin^2(\theta'-\alpha)\sin^2\varphi\right]+2\beta\sin\alpha\cos\varphi\sin(\theta'-\alpha)\right)\Big] - PartC$$

(1)

Where $h = b/cos\alpha$, $\omega = 2\pi c/\lambda$ and the following notation is introduced:

$$P = \cos\alpha - \beta\sqrt{\varepsilon-\sin^2(\theta'-\alpha)} + i\gamma^{-1}K\sin\alpha$$

$$\Sigma = \sin\alpha + \beta\cos\varphi\sin(\theta'-\alpha) - i\gamma^{-1}K\cos\alpha \qquad (2)$$

$$K = \sqrt{1+(\gamma\beta\sin(\theta'-\alpha)\sin\varphi)^2}$$

The target is rotated relative to the trajectory of the bunch of charged particles and so the relation between the angle of particle flight, the wedge angle, and the angle of rotation of the target can be written as $\alpha = \pi/2 - \psi - \phi$. The spectral angular distribution of polarised radiation is obtained by attaching the angles of observation to the Cartesian system of coordinates related to the target as in figure 2. The system of observation can be passed from the trajectory of a charged particle providing the relation between angles θ and θ' which can be written as $\theta = \theta' - \alpha$. The full derivation for equation 1 can be found in the paper [1].

4 The Code

I chose to use **C++** due to it's speed and efficiency in doing large calculations. Furthermore, since the equation deals with imaginary numbers, a framework for calculations in i is necessary. In C++ such a standard exists and using C++ 17 we have multiple functions for their manipulation. This allows us to simply focus on copying the equation into code. The appendix shows the full code but the understanding of it is explained here.

For creating an application or a more developed program I would recommend using classes; however as a proof of concept, classes and standard use of C++ is not entirely necessary. A functional approach with global variables that can be accessed with pointers is simple and clear to understand for a single **main.cpp** file and is sufficient for our use case. The simplicity of this approach provides a much clearer understanding of the equation used in the context of the code and makes it very easy to modify or adjust for a variety of simulations. For compilation **CMake** is used for easy cross platform compiling and setup for potential use in applications.

The basic idea is to have a single file with global variables that can be accessed by functions that are added and multiplied as in equation (1). The parts labelled A, B and C in equation (1) correspond to the functions *PartA(), PartB() and PartC()* that are then further split to make it easier to write and understand the code as can be seen here:

```
long double SpectralAD()
{
    /*returns spectral angular distribution split into
        ↪ parts from eq 16:
      - A is first part of eq, B is inner brackets
          ↪ part, C is last part of eq.
      - Numbered in chunks.
    */
    return PartA() * PartB() * PartC();
}
```

The corresponding functions are then further split into smaller chunks to make it more legible and clearer to understand the flow of the program. We also globally declare the constants to be used from equation (1):

```
const complex<long double> i(0, 1.0);
const long double pi = acos(-1.0);
// variables

const long double a = 0.045;
const long double b = 0.015;

const long double gamma1 = 12.0;
const long double permitivity = 1.41 * 1.41;
const long double phi = pi / 4.0;
long double phiVar = 0;

long double psi = 0.0 * pi / 180.0;

const long double lambda = 0.004;
const long double c = 299792458.0;
const long double Beta = sqrt(1.0 - (1.0 / (gamma1 *
    ↪ gamma1)));
const long double omega = (2.0 * pi * c) / lambda;
const long double alpha = (pi / 2.0) - phi - psi;

//angle
long double theta_p = -20.0 * pi / 180.0;
long double *ptr_theta_p = &theta_p;

long double h = b / cos(alpha);
```

where *phiVar* is the azimuthal angle. Long double is used for the declaration of variables throughout and all angles are expressed in radians. Such precision is not necessary as the accuracy of double is easily sufficient for providing the correct results. However, it does not make any substantial difference to the running of the executable and is just as fast. Converting to the use of double over long double can easily be done if needed.

The complex library in C++ 17 includes the ability to declare a complex number (long double)

```
const complex<long double> i(0, 1.0);
```

which corresponds mathematically to $z = 0 + 1.0i$ and so we have declared the constant i for use within our simulation. We can simply multiply the constant i wherever it is necessary in the equation to ease the declaration of our complex functions. In equation (1) we frequently take the magnitude squared $|z|^2$; in C++ 17 the *norm()* function that is part of the complex library does just this and so we can use it to evaluate the solution.

By using a pointer to *theta_p* we can change the angle input for our simulation for the whole code as we are adjusting the value directly held in memory removing the need to pass the value on as an input to the functions. This can be done in the **main()** function where we also call our **SpectralAD()** at the angle that is set and output it to a file *data.txt* along with the angle. To produce data for our spectral angular distribution we can choose 5000 evenly spaced iterations of the simulation:

```
int main()
{
    // Outputs spectral angular distribution to data.
        ↪ txt at regular intervals of granularity
        ↪ from -20 to 70 deg.
    const int granularity = 5000;
    ofstream myfile;
    myfile.open("data.txt");
    for (int i = 0; i < granularity; i++)
    {
        myfile << theta_p * 180 / pi << " " <<
            ↪ SpectralAD() << endl;
        *ptr_theta_p += 90 * pi / (180 * granularity);
    }
    myfile.close();

    return 0;
}
```

We can see how simple it is to iterate using the pointer to *theta_p* to globally change the angle of the charged particle. 5000 evenly spaced iterations is more than enough for a smooth curve with almost instant execution. The output of the data is a text file **data.txt** with two columns, the first corresponding to the angle and the second to the polarisation radiation. Now we can use a plotting library of any choice to visualise the data.

5 Simulation

To assess whether the code works I consistently compared my output graph to Fig.4 in [1] using the same parameters.

The paper [1] mentions that the azimuthal angle φ is to be 0 when $\theta' > \alpha$ and π otherwise. I have found that this is not the case - plotting our initial test plot I found that φ must always be 0 to work. A valid assumption would be that the paper [1] uses the wrong axis or there has been a misinterpretation of the equation. In figure 4 we analyse this unusual error further.

We can use **Python** due its wide range of libraries for data science and visualisation. We can easily load in the data from **data.txt** and produce plots by loading the column into a **Numpy** array.

```
c_s = 299792458

x = np.loadtxt('build/data.txt', usecols=(0), unpack=
    ↪ True)
y = np.loadtxt('build/data.txt', usecols=(1), unpack=
    ↪ True)

# plt.plot(x, y)
plt.plot(x, y/(math.exp(2)/(math.pi**2 * c_s)))
```

We use a factor of $\frac{e^2}{\pi^2 c}$ for a direct comparison to Fig.4 in [1], obtaining Fig.3.

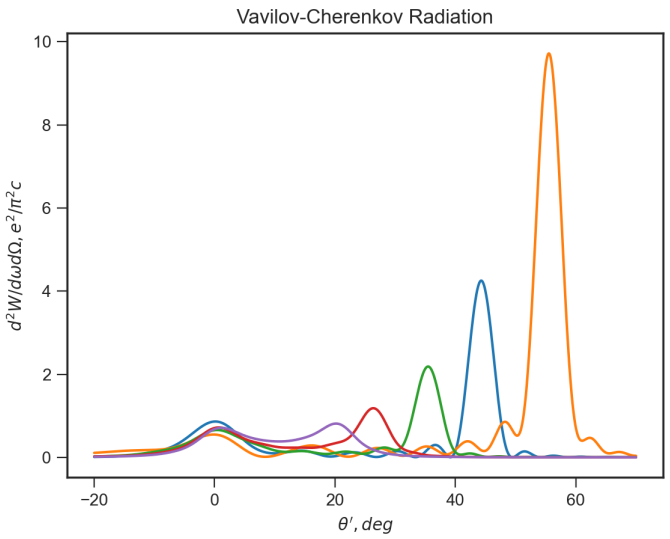

Figure 3: Angular distribution of polarisation radiation generated by a charged particle with energy $\gamma = 12$ moving near a dielectric wedge. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, $a = 45mm$, and $\lambda = 4mm$

Where Ψ is -10 (orange), 0 (blue), 10 (green), 25 (red), 45 (violet) in degrees. We can see strong peaks representing VCR and small peaks at 0 degrees for all results for DR as expected. Small wavelets are also clearly visible for all but 25 and 45 degree angles of attack, which appear to have an interplay between DR and VCR. This plot is identical to Fig.4 in [1] showing that the code works.

As this only worked when $\varphi = 0$, I also made a 3D plot to see the relationship between the spectral angular distribution and azimuthal angle to understand the unusual behaviour:

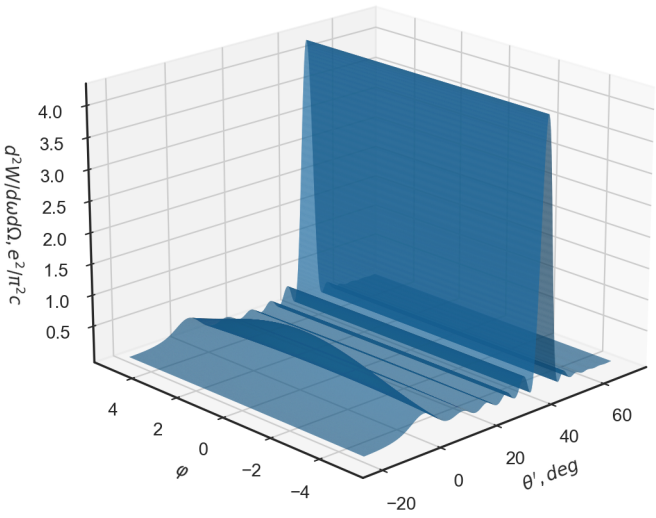

Figure 4: Angular distribution of polarisation radiation generated by a charged particle with energy $\gamma = 12$ moving near a dielectric wedge against azimuthal angle. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, $a = 45mm$, and $\lambda = 4mm$

Between -5 and 5 degrees of φ we find that DR falls off as is expected [6]; however the VCR peak stays completely constant where it should also fall, suggesting a possible error in the code or a misinterpretation of equation (1). It is very unusual that φ must be 0 to obtain an analogous plot 3 to Fig.4 in [1] - I have not completely assessed this so further analysis is required.

5.1 Arbitrary Simulations

Using the same techniques as in sections 4 and 5 we can output any of the variables in our equation to analyse various relationships between the variables and further understand polarisation radiation; it is also important to further show that this simulation works. We can analyse the dependence on the distance of the charged particle trajectory from the dielectric target at 0 degrees on a 3D plot by changing the value of 'a' using a pointer as described similarly in section 4 for **theta_p**:

```
// 3D plot
    granularity = 1000;
    *ptr_theta_p = -20.0 * pi / 180;
    ofstream dfile;
    dfile.open("data3D.txt", ios::app);
    *ptr_a = 0.01;
    for (int j = 0; j < granularity; j++)
    {
        for (int i = 0; i < granularity; i++)
        {
            dfile << theta_p * 180 / pi << "␣" <<
                ↪ SpectralAD() << "␣" << a << endl;
            *ptr_theta_p += 90 * pi / (180 *
                ↪ granularity);
        }
        *ptr_a += 0.035 / granularity;
        *ptr_theta_p = -20.0 * pi / 180;
    }
    dfile.close();
```

The granularity is turned down to 1000 iterations to speed up the simulation as there will now be 1000^2 iterations as we are plotting the distribution against 2 variables. Loading in the **data3D.txt** into **Python** using the **tri_surf()** function that is part of Axes3D from mpl_toolkits.mplot3d

```
''
# 3D plot
x = np.loadtxt('build/data3D.txt', usecols=(0),
    ↪ unpack=True)
z = np.loadtxt('build/data3D.txt', usecols=(1),
    ↪ unpack=True)
y = np.loadtxt('build/data3D.txt', usecols=(2),
    ↪ unpack=True)

fig = plt.figure()
ax = Axes3D(fig)

ax.plot_trisurf(x, y, z/(math.exp(2)/(math.pi**2 *
    ↪ c_s)))

# plt.yscale("log")
ax.set_xlabel(r'$\theta^\prime,␣deg$')
ax.set_zlabel(r'$d^2W␣/␣d\omega␣d\Omega,␣e^2␣/␣\pi^2
    ↪ c$')
ax.set_ylabel(r'$a,␣m$')
```

```
ax.set_title('Vavilov-Cherenkov Radiation 3D plot')
'''
```

we obtain the plot that is seen in figure 5.

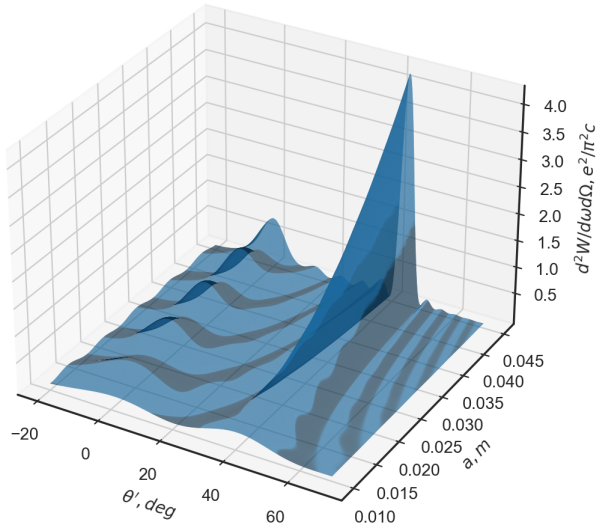

Figure 5: Angular distribution of polarisation radiation generated by a charged particle with energy $\gamma = 12$ moving near a dielectric wedge. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, and $\lambda = 4mm$.

As the parameter 'a' and so the size of the prismatic target increases, the polarisation radiation also increases. To further understand the rate of increase of polarisation radiation as 'a' increases I also obtained sections of the plot at $\theta' = 45^o$ and $\theta' = 0^o$:

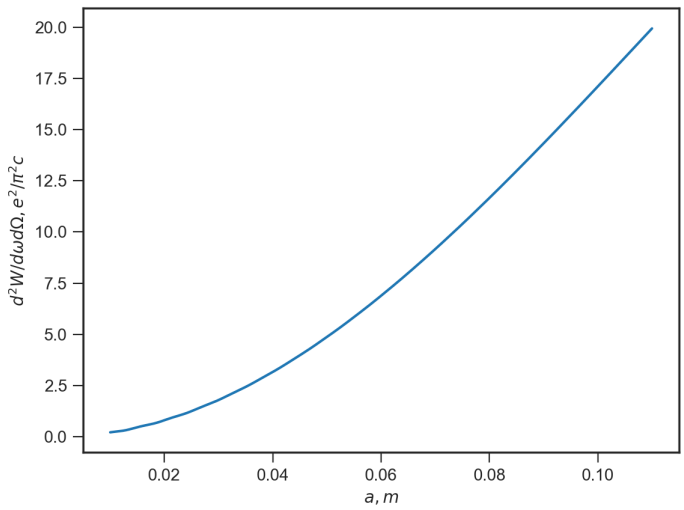

Figure 6: VCR Polarisation radiation dependence on distance 'a' from target generated by a charged particle with energy $\gamma = 12$ moving near a dielectric wedge. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, $\theta' = 45^o$, and $\lambda = 4mm$.

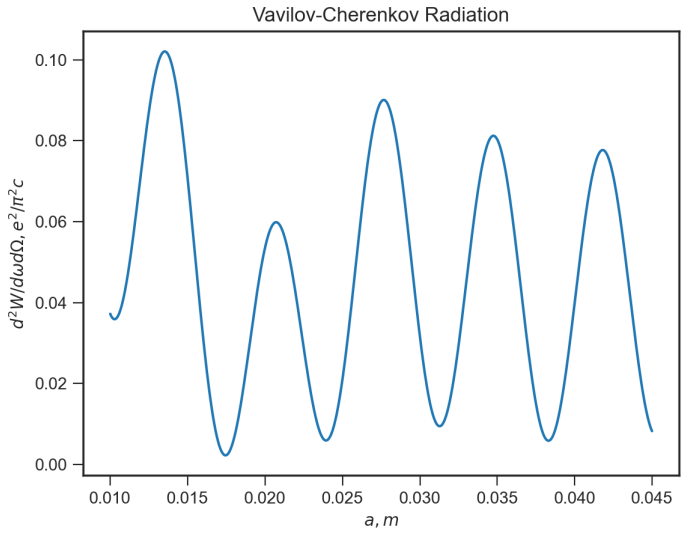

Figure 7: DR Polarisation radiation dependence on distance 'a' from target generated by a charged particle with energy $\gamma = 12$ moving near a dielectric wedge. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, $\theta' = 45^o$, and $\lambda = 4mm$.

We can see that polarisation radiation at 45^o, corresponding to VCR, increases quadratically with increasing size 'a' of the prismatic target as is expected [6]; the corresponding diffraction radiation oscillates at regular intervals as the phase of DR radiation changes when the dielectric size changes.

5.2 Diamond Light Source Simulation

As noted in the introduction, the generation and measurement of Vavilov-Cherenkov Radiation is incredibly useful in the detection of particles in astronomy and particle physics, and also in synchrotrons for the analysis of atomic structures such as at Diamond Light Source. The understanding and visualisation of polarisation radiation can lead to better techniques such as the development of beam position monitors for high-energy electron machines [7]. Changing parameters to analyse expected distributions are much less costly and simpler if we can simulate it.

Diamond Light Source (DLS) in the UK is a 3GeV synchrotron that is used as a giant microscope, harnessing the phenomenon of Vavilov-Cherenkov Radiation that scientists can use to study anything from fossils to jet engines to viruses and vaccines [8]. In order to simulate the radiation output of DLS we use the relationship in equation (3) to obtain γ.

$$\gamma = \frac{E_{total}}{E_{rest}} = \frac{3 * 10^3 MeV}{0.511 MeV} \tag{3}$$

Keeping the same constants as in figure 3 at $\Psi = 0$ the Spectral Angular Distribution is:

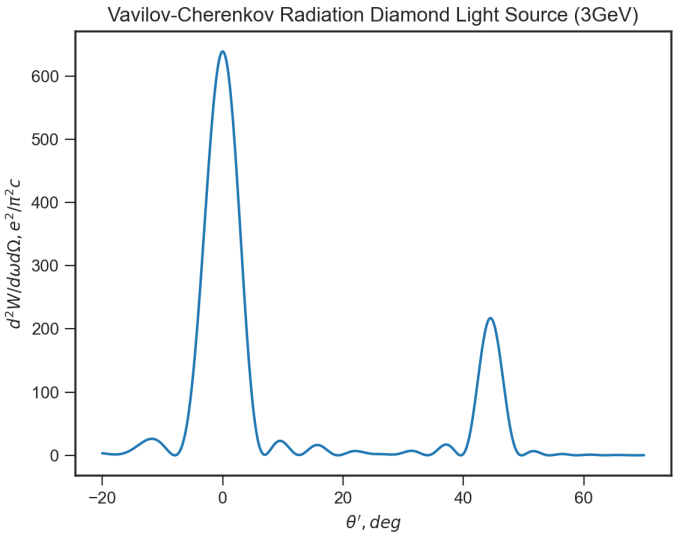

Figure 8: Angular distribution of polarisation radiation generated by a charged particle with energy $\gamma = 5870.84148728$ moving near a dielectric wedge simulating Diamond Light Source. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, $a = 45mm$, and $\lambda = 4mm$.

We can see that the simulated results at these parameters for Diamond Light Source produce strong DR at 0 degrees and relatively weaker VCR at ∼45 degrees. Comparing to our initial $\gamma = 12$ charged particle we find that there is also a 10 order of magnitude increase in radiation. At higher energies there is a more radiation and diffraction radiation is more pronounced.

These results do appear to be very wide and a more realistic target may be one where $a = 1cm$. From section 5.1 we can already predict that the polarisation radiation will be much larger than in figure 8 as it would increase as the parameter 'a' increases.

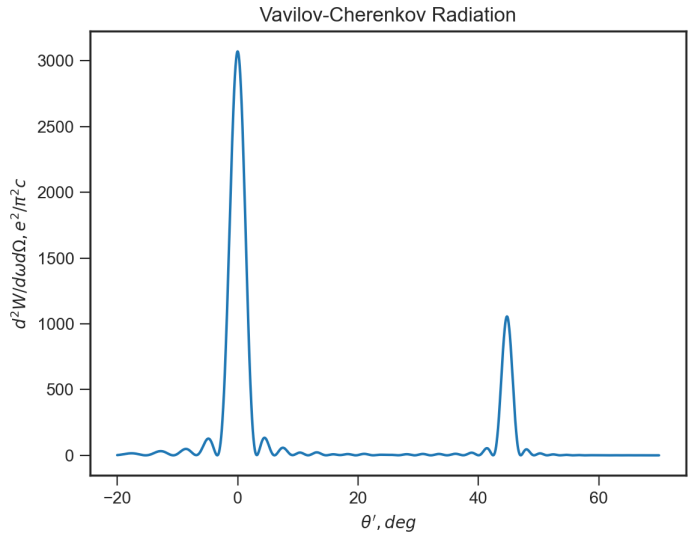

Figure 9: Angular distribution of polarisation radiation generated by a charged particle with energy $\gamma = 5870.84148728$ moving near a dielectric wedge simulating Diamond Light Source. Parameters are $\epsilon = 1.41$, $b = 15mm$, $\phi = \pi/4$, $a = 1cm$, and $\lambda = 4mm$.

From figure 9 we can confirm the greater polarisation radiation and we can also see that the peeks are a lot narrower when we use a "wider" target where the parameter 'a' is 1cm. These simulations have been successful in showing that the code works as intended, but to verify these results practical experiments should be done at DLS to compare and test the effectiveness of this code for the simulation of Vavilov-Cherenkov radiation; the results seen in figures 8 and 9 confirm the simulation provides the expected results [6] and so using this code for developing novel beam position monitors would be of particular use for research and analysis.

6 Conclusion

Vavilov-Cherenkov Radiation is an important phenomena in multiple fields of research in astronomy and particle physics for the development of detection and analysis of particles and materials in synchrotrons. Hence we have developed code in **C++** and **Python** to simulate polarisation radiation using the generalised equation developed by M. V. Shevelev and A. S. Konkov [1]. The simulations have reproduced Fig.4 from [1] although the parameter φ was always set to 0; this may be due to misinterpretation of the paper [1] or an error in the code which should be further analysed. The code has matched the expected [6] quadratic increases in polarisation radiation with increasing size 'a' of the prismatic target. Finally we simulated polarisation radiation with an energy of 3GeV for Diamond Light Source, which is important for assisting particle and material analysis, and found that polarisation radiation increases greatly and there is a greater peak of DR. We also found that a larger target is required with larger energies to provide stronger, more refined, VCR and DR.

This code uses a very functional approach and for implementation into larger applications a class based approach would be more optimal; however it is very simple to read and modify for use in future projects and provides many variables that can easily be analysed. Future experimental confirmation of the code at Diamond Light Source can also be useful, and further development of the simulation would assist in multiple fields including astronomy and particle physics research, being particularly paramount to the production of novel beam position monitors[8][7].

References

[1] M.V.Shevelev and A.S.Konkov, *Peculiarities of the Generation of Vavilov–Cherenkov Radiation Induced by a Charged Particle Moving Past a Dielectric Target*, 13 May 2013.

[2] R.Kieffer et al, *Direct Observation of Incoherent Cherenkov Diffraction Radiation in the Visible Range*, 3 April 2018.

[3] V. L. Ginzburg and I. M. Frank, Dokl. Akad. Nauk SSSR 56, 699, 1947

[4] K. Sakaue, M. Brameld et al, *Investigation of the Coherent Cherenkov Radiation Using Tilted Electron Bunch*, 2017

[5] J V Jelley Br. J. Appl. Phys. 6 227, *Cerenkov radiation and its applications*, 1955

[6] I. E. Tamm, *General Characteristics of Vavilov-Cherenkov Radiation*, New Series, Vol. 131, No. 3395, pp. 206-210, 22 January 1960

[7] P. Forck, P. Kowina, and D. Liakin, *Beam position monitors* https://core.ac.uk/download/pdf/44233904.pdf

[8] Diamond Light Source, *About Us* https://www.diamond.ac.uk/Home/About.html

Appendices

* The full source code and cmake files and plots are available on my personal Github

* C++ source code **main.cpp**

```
#include <math.h>
#include <iostream>
#include <fstream>
#include <complex>

using namespace std;

const complex<long double> i(0, 1.0);
const long double pi = acos(-1.0);
// variables

long double a = 0.1;
long double *ptr_a = &a;
long double b = 0.015;
long double *ptr_b = &b;

//const long double gamma1 = 12;
const long double gamma1 = 5870.84148728;
const long double permitivity = 1.41 * 1.41;
long double phi = pi / 4.0;
long double *ptr_phi = &phi;
long double phiVar = 0;
long double *ptr_phiVar = &phiVar;

long double psi = 0.0 * pi / 180.0;

const long double lambda = 0.004;
const long double c = 299792458.0;
const long double Beta = sqrt(1.0 - (1.0 / (gamma1 *
    ↪ gamma1)));
const long double omega = (2.0 * pi * c) / lambda;
const long double alpha = (pi / 2.0) - phi - psi;

//angle
long double theta_p = -20.0 * pi / 180.0;
long double *ptr_theta_p = &theta_p;

long double h = b / cos(alpha);

long double Kval()
{
    return sqrt(1.0L + pow(gamma1 * Beta * sin(theta_p
        ↪  - alpha) * sin(phiVar), 2.0L));
}

complex<long double> Pval()
{
    return cos(alpha) - Beta * sqrt(permitivity - pow(
        ↪ sin(theta_p - alpha), 2.0L)) + i * pow(
        ↪ gamma1, -1.0L) * Kval() * sin(alpha);
}

complex<long double> SigmaVal()
{
    return sin(alpha) + Beta * cos(phiVar) * sin(
        ↪ theta_p - alpha) - i * pow(gamma1, -1.0L) *
        ↪  Kval() * cos(alpha);
}

long double PartA1()
{
    return ((exp(2.0L) * pow(Beta, 2.0L)) / (4.0L *
        ↪ pow(pi, 2.0L) * c)) * (pow(cos(theta_p -
        ↪ alpha), 2.0L) / norm(Pval())) * norm((
        ↪ permitivity - 1.0L) / permitivity);
}

complex<long double> PartA2()
{
    return 1.0L - ((Pval() * exp(i * (omega / (Beta *
        ↪ c)) * SigmaVal() * a * (1.0L / tan(phi))) +
        ↪  SigmaVal() * (1.0L / tan(phi)) * exp(-i *
        ↪ a * (omega / (Beta * c)) * Pval())) / (Pval
        ↪ () + SigmaVal() * (1.0L / tan(phi))));
}

long double PartA()
{
    return PartA1() * norm(PartA2());
}

complex<long double> PartB1()
{
    return permitivity / (permitivity * cos(theta_p -
        ↪ alpha) + sqrt(permitivity - pow(sin(theta_p
        ↪  - alpha), 2.0L)));
}

complex<long double> PartB2()
{
    complex<long double> p1 = cos(alpha) * (pow(gamma1
        ↪ , -1.0L) * sin(theta_p - alpha) - i * Kval
        ↪ () * cos(phiVar) * sqrt(permitivity - pow(
        ↪ sin(theta_p - alpha), 2.0L)));
    complex<long double> p2 = sin(alpha) * (i * Kval()
        ↪  * sin(theta_p - alpha) + pow(gamma1, -1.0L
        ↪ ) * cos(phiVar) * sqrt(permitivity - pow(
        ↪ sin(theta_p - alpha), 2.0L)));
    complex<long double> p3 = gamma1 * Beta * sin(
        ↪ theta_p - alpha) * sqrt(permitivity - pow(
        ↪ sin(theta_p - alpha), 2.0L)) * pow(sin(
        ↪ phiVar), 2.0L);
    return p1 + p2 - p3;
}

long double PartB3()
{
    long double p1 = norm(sqrt(permitivity) / (cos(
        ↪ theta_p - alpha) + sqrt(permitivity - pow(
        ↪ sin(theta_p - alpha), 2.0L))));
    long double p2 = pow(gamma1 * sin(phiVar), 2.0L) *
```

```
        ↪  (pow(sin(theta_p - alpha), 2.0L) + norm(
        ↪ sqrt(permitivity - pow(sin(theta_p - alpha)
        ↪ , 2.0L))));
    return p1 * p2;
}

long double PartB4()
{
    return 1.0L - pow(Beta, 2.0L) * pow(cos(theta_p -
        ↪ alpha), 2.0L) + 2.0L * Beta * pow(gamma1,
        ↪ -2.0L) * sin(alpha) * sin(theta_p - alpha)
        ↪ * cos(phiVar) - pow(gamma1, -2.0L) * pow(
        ↪ sin(alpha), 2.0L) * (pow(Kval(), 2.0L) -
        ↪ pow(gamma1, -2.0L));
}

long double PartB()
{
    return norm(PartB1()) * norm(PartB2()) + PartB3()
        ↪ * PartB4();
}

long double PartC1()
{
    return exp(-2.0L * (omega / (gamma1 * Beta * c)) *
        ↪  (h + a * 1.0L / tan(phi)) * Kval() * cos(
        ↪ alpha));
}

long double PartC2()
{
    return pow(Kval(), 2.0L) * (1.0L - pow(Beta, 2.0L)
        ↪  * pow(cos(theta_p - alpha), 2.0L) + pow(
        ↪ Beta, 2.0L) * pow(sin(alpha), 2.0L) * (1.0L
        ↪  - pow(sin(theta_p - alpha), 2.0L) * pow(
        ↪ sin(phiVar), 2.0L)) + 2.0L * Beta * sin(
        ↪ alpha) * cos(phiVar) * sin(theta_p - alpha)
        ↪ );
}

long double PartC()
{
    return PartC1() / PartC2();
}

long double SpectralAD()
{
    /*returns spectral angular distribution split into
        ↪  parts from eq 16:
      - A is first part of eq, B is inner brackets
          ↪ part, C is last part of eq.
      - Numbered in chunks.
    */
    return PartA() * PartB() * PartC();
}

int main()
{
    // Outputs spectral angular distribution to data.
        ↪ txt at regular intervals of granularity
        ↪ from -20 to 70 deg.
    int granularity = 5000;
    ofstream myfile;
    myfile.open("data.txt");

    for (int i = 0; i < granularity; i++)
    {
        myfile << theta_p * 180 / pi << " " <<
            ↪ SpectralAD() << endl;
        *ptr_theta_p += 90 * pi / (180 * granularity);
    }

    myfile.close();

    // 3D plot
    granularity = 1000;
    *ptr_theta_p = -20.0 * pi / 180;
    ofstream dfile;
    dfile.open("data3D.txt", ios::app);
    *ptr_a = 0.01;
    for (int j = 0; j < granularity; j++)
    {
        for (int i = 0; i < granularity; i++)
        {
            dfile << theta_p * 180 / pi << " " <<
                ↪ SpectralAD() << " " << a << endl;
            *ptr_theta_p += 90 * pi / (180 *
                ↪ granularity);
        }
        *ptr_a += 0.035 / granularity;
        *ptr_theta_p = -20.0 * pi / 180;
    }
    dfile.close();
    return 0;
}
```

* Basic Python script used for plotting **Plot.py**

```
import matplotlib.pyplot as plt
import numpy as np
import math
from numpy.core.function_base import linspace
plt.style.use('seaborn-ticks')

c_s = 299792458

x = np.loadtxt('build/data.txt', usecols=(0), unpack=
    ↪ True)
y = np.loadtxt('build/data.txt', usecols=(1), unpack=
    ↪ True)

# plt.plot(x, y)
plt.plot(x, y/(math.exp(2)/(math.pi**2 * c_s)))

# plt.yscale("log")
plt.xlabel(r'$\theta^\prime,␣deg$')
plt.ylabel(r'$d^2W␣/␣d\omega␣d\Omega,␣e^2␣/␣\pi^2c$')
plt.title('Vavilov-Cherenkov␣Radiation')

'''
# 3D plot
x = np.loadtxt('build/data3D.txt', usecols=(0),
    ↪ unpack=True)
z = np.loadtxt('build/data3D.txt', usecols=(1),
    ↪ unpack=True)
```

```
y = np.loadtxt('build/data3D.txt', usecols=(2),
    ↪ unpack=True)

fig = plt.figure()
ax = Axes3D(fig)

ax.plot_trisurf(x, y, z/(math.exp(2)/(math.pi**2 *
    ↪ c_s)))

# plt.yscale("log")
ax.set_xlabel(r'$\theta^\prime, deg$')
ax.set_zlabel(r'$d^2W / d\omega d\Omega, e^2 / \pi^2
    ↪ c$')
ax.set_ylabel(r'$a, m$')
ax.set_title('Vavilov-Cherenkov Radiation 3D plot')
'''

plt.show()
```