Simulation of XRD analysis on crystalline materials

A. Back

(Dated: 15 July 2021)

I. INTRODUCTION:

XRD or X-Ray Powder diffraction involves analyzing the scattering patterns of microwaves off of crystalline materials in order to gain insight to the structure of the material.[1] The process involves the movement of an emitter and a detector. The emitter sends out microwaves towards a powdered sample of the material of interest. The microwaves then scatter off of the atoms in the structure and the resulting microwaves are caught by the detector.

A. Usage

XRD analysis can serve many uses in fields of science such as physics, chemistry, and geology. One major use of XRD is to identify the phase or structure of a material.[1] In some cases XRD is used to identify an unknown material by comparing the unknown materials XRD pattern to know patterns.[2]

B. Project Goals

The purpose of this project is to use GlowScript to simulate the process of XRD and to use this simulation to produce graphs comparable to those we get from real XRD analysis. Known XRD patterns will be used to test the accuracy of the simulation and the polonium crystalline structure will be used. Previous simulations of XRD have been conducted[3], however these previous simulations involve some approximations and are applied to large sample sizes. The simulation in this experiment hopes to simulate the interactions in a more direct manner which will require a decrease in the sample size of materials in the simulation.

II. THEORY:

XRD is seen as a result of Bragg's law.[4] Braggs law shows how waves will reflect off of crystaline materials. It is given as the equation

$$n\lambda = 4d\sin(\theta), \tag{1}$$

where n is a positive integer, λ is the wavelength of the light, d is the distance between lattices, and θ is the angle of reflection.

A geometric derivation of Bragg's Law follows that will utilize figure 1 above.[5] This photo shows a wave of monochromatic light being reflected in a crystal structure.

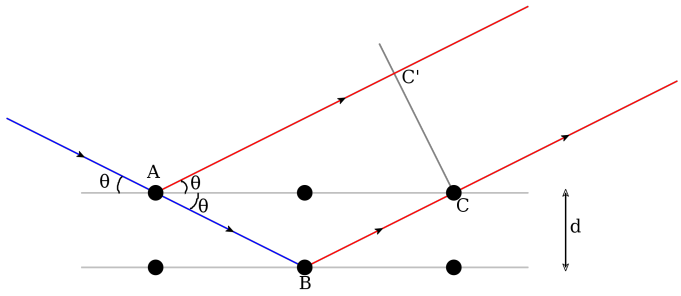

FIG. 1. Model of Bragg's Law in a crystal structure

A. Bragg's Law Derivation

The derivation shown is modified from Schields et. al.[6] To begin we need an equation for the path difference between the ray AC' and BC. This path difference is (AB + BC) - (AC'). In order to have the desired constructive interference, we will the path difference to be an integer multiple of the wavelength. This will yield the equation

$$(AB+BC)-(AC') = n\lambda. \tag{2}$$

We can see from figure 1 that AB = BC = $\frac{d}{\sin(\theta)}$ and AC = $\frac{2d}{\tan(\theta)}$, which means that AC' = AC$\cdot\cos(\theta)$ = $\frac{2d}{\tan(\theta)}\cos(\theta)$ = $\frac{2d}{\sin(\theta)}\cos^2(\theta)$. Putting everything together we get that

$$n\lambda = \frac{2d}{\sin(\theta)} - \frac{2d}{\sin(\theta)}\cos^2(\theta) = \frac{2d}{\sin(\theta)}(1-\cos^2(\theta)) = 2d\sin(\theta), \tag{3}$$

which is our expected result for Bragg's Law.

B. Bragg's Law for XRD

Both the emitter and detector move through an angle of 2θ. The resulting amplitudes of the microwaves are then graphed vs 2θ as seen in figure 1. The resulting graph should show areas of no or low amplitude with large spike in amplitude at certain locations.[1] These peaks appear due to Bragg's scattering, wherein, the scattered microwaves will align constructively at certain values of θ while for the rest of the values, the destructive interference will cause the low values of amplitude.[6]

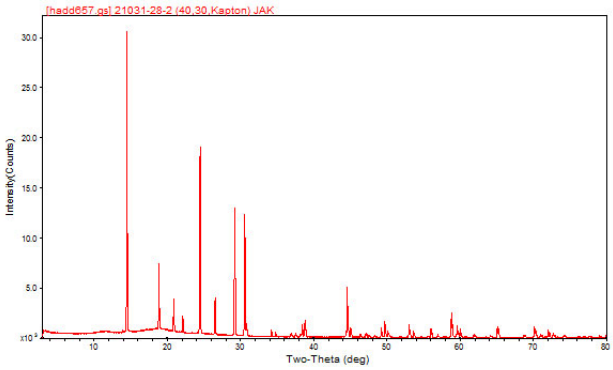

FIG. 2. Example of XRD diffraction pattern[7]

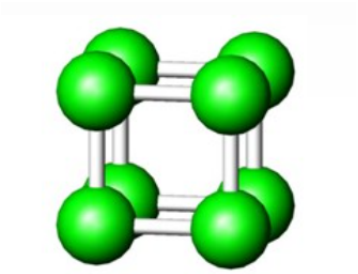

FIG. 3. Example of Po Crystal Structure[8]

C. Crystal Structure of Po

Figure 3 shows a model of Polonium from Charles Maxwell.[8] Polonium was used as the crystal of choice for this experiment because it has a simple cubic structure which allows for easy modeling in GlowScript. The distance between each Po atom is 3.34 angstroms.[8] The radius of each Po is 1.45 angstroms.[9]

III. METHODS:

This simulation is programmed in the GlowScript VPython IDE. The code has four main components; these components are the light wave code, atomic structure model code, reflection code, and graphing code. All four parts are integrated together into the final simulation. The length unit used to keep models scaled throughout the code is angstroms. The full code can be found in the appendix.

A. Light Wave Code

The wavelength used in this simulation is .8 angstrom, which is the typical wavelength used in traditional XRD[10]. The wave was created by making a curve in GlowScript and appending on points to the curve with the points being mapped from a cosine function. Since a normal cosine function move horizontally, the following transformations were used:

$$\begin{aligned} x' &= x\cos(\theta) - y\sin(\theta \\ y' &= x\sin(\theta) + y\cos(\theta), \end{aligned} \tag{4}$$

where θ is angle you want to rotate your axis. Using equation 4 and the basic cosine graphing rules learned in precalculus, a wave can be created at any point in space moving at any specified angle.

B. Atomic Structure Code

Using a triple for-loop moving through the x, y, and z axis in increments of the bond length. Each Po atom was created and added to a list of all atoms within the crystal model. The code allows a user to specify how wide, in atoms, they want the crystal to be and so the total number of atoms in the structure is equal to the width inputted cubed.

C. Reflection Code

To simulate the reflection of waves off of the atomic structures, there are two reflections that will come off of a particle. When a wave hits it will continue straight through on its path (example is AB in figure 1) and another wave will be created that will reflect off towards the detector. This simplification of the reflection is made since on atomic reflections the light will go off in all directions. This would require too many resources to be feasible in the Glowscript environment, thus, only the wave that continues through and the wave that hits the detector are simulated. This should not throw off results too much since all other reflected waves will not hit the detector.

D. Graphing Code

In order to graph the simulated data, the average difference between the un-transformed y-values of all waves that hit the detector are taken and if they are above a certain amount the program will graph it as a zero and if it is below the threshold it is graphed as a the average difference squared.

E. Putting it all Together

In the final program, the program begins by creating the crystal model the same as it does in section B. Once the crystal is created the reflection code from section C is run through angles from 10 to 22 degrees in increments of .5 theta. The reflection code also loops through emitter surface sending waves across the entirety of the emitter plate with waves being separated by two times the wavelength. For every angle the graphing code from section D is run for that angles data. This process is repeated for every angle in the loop to get a graph that should be similar to that of experimental results.

IV. RESULTS:

The graphs produced by the simulation showed evidence that the simulation is successfully detecting angles of constructive interference for the polonium crystal model. The graphs did however have some spikes in areas that should have been destructively interfering. The limited memory capabilities of GlowScript, however, did not make it possible to test the simulation with a larger crystal size that might eliminate the inconsistencies between the expected and actual graphs of the polonium crystal.

A. Initial Runs

Due to the fact that, for security reasons, browsers only allow 4 gigabytes of memory to be allocated to any one tab, the simulation could only be run with a crystal sample containing 125 atoms. This is much less than the preferred sample size of a few thousand atoms.The expected result of this is that we should see shorter peaks and possible unexpected bumps in the data. The simulation takes approximately 30 seconds to run at one angle. In order to get graphed results and the simulation must be run at multiple angles. It was found that the simulation could be run for 15 iterations before crashing due to reaching the memory limit. This resulted in simulations being run over samples of two to three degrees in order to have sample points spaced close enough to get an accurate graph. The simulation was ran over points that are expected to have peaks according to the graph below of an actual XRD scan of polonium.

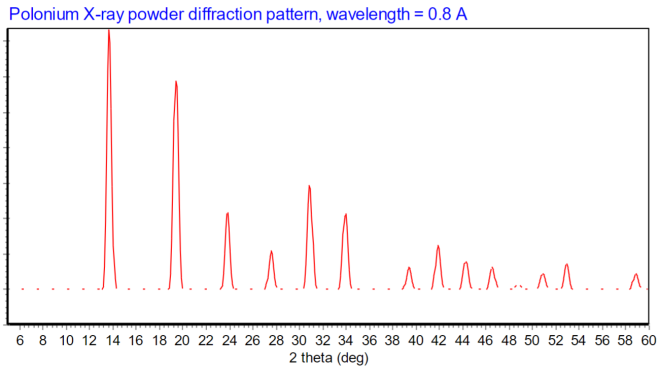

FIG. 4. Polonium XRD Diffraction Pattern[8]

A sample output of the program from 18 to 20 degrees is shown in figure 5. The rest of the outputs can be found in appendix B.

B. Comparison to Experimental Results

Comparing the graphs in appendix B to the graph in figure 4 we can see that locations with large bumps, such as at 14 and 20 degrees, do appear on our simulation graphs. The expected and simulated graphs do begin differ as we move away from these larger spikes. Points appear on the simulation graph of peaks where there should not be peaks. These could be caused by the fact that there are only 125 atoms and so there is not enough interference to cause full destructive interference. The lower number of atoms is also a possible cause for the graphs becoming less accurate at points with smaller expected bumps. Since there is less reflections, there would be less constructive interference to create the peaks.

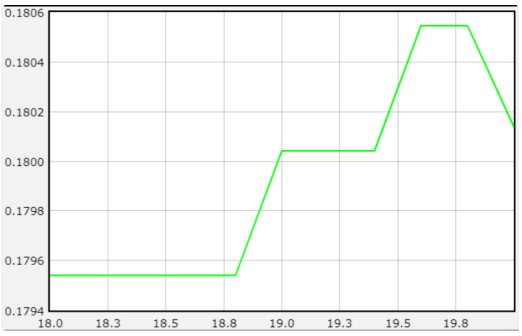

FIG. 5. Simulation Graph from 18-20 Degrees (15 iterations)

V. CONCLUSION AND FURTHER STUDY:

Overall, the simulation appears to show some evidence that with more time and processing power, an accurate graph could be produced. The are still changes that need to be made, both in the environment the program runs in and algorithmic changes.

A. Run-time and Further Changes

The run-time of the simulation is currently not a primary issue although it could lead to problems in the future. Improvements in run-time of the program would be useful for allowing the program to create more accurate graphs with more precision with angle measurements.

The most important future change for this project would be to move the simulation to a different programming environment that would allow for better RAM usage and multi-threading. The biggest limiting factor in this simulation was that it was limited to only 4GB of RAM. More memory usage would allow for more atoms in the crystal sample and more precise graphing. The multi-threading would allow for multiple waves to run at any one time greatly decreasing the run-time of the simulation.

Another possible option would be to run the program on a computer with more RAM space. Because the simulation holds so much data at any one time as the program runs more resources are required and so a computer with increased RAM would allow for large and more complex simulations.

B. Algorithm Changes and Additions

There are two algorithm changes I suggest to improve the program in the future. The first is that currently the simulation checks for particle collisions for every point that is added to the wave. However, this is overkill and could be reduced by adding a system to the code that would only check for collisions in a less frequent amount.

The second change would be to create an algorithm that would make it so instead of checking every atom in the list, it would only check a subsection of the list that you know has a possibility of collisions. This would most likely require sorting the atoms list by position.

VI. APPENDIX:

A. Simulation Code

```
##############Crystal Code########################
#bond-length
bl=3.345
#radius of polonium
r=1.45
#lattice size (#of atoms across)
ls=5
#length of each side of lattice in angstroms
len=ls*bl
#list to hold all atoms
atoms = []
for X in range(-len/2,len/2,bl):
    for Y in range(-len/2,len/2,bl):
        for Z in range(-len/2,len/2,bl):
            #create sphere and add to list of atoms
            atoms.append(sphere(pos =
                vec(X,Y,Z),radius=r,color=color.blue))
```

```
##############Reflection Code########################
#list to hold ending y-values
        Yvals = []
        for Z in range(-len+bl,len-bl,3):
            for Y in range(-len+bl,len-bl,3):
                #Reflection Code
                theta = radians(b)
                #initial x value (not rotated)
                X = -100
                Xinit = X
                #change in x
                deltaX = .01
                #the starting y value
                initY = Yprime(X,theta,Y)
                #create emitter plate
                emitter =
                    box(pos=vec(Xprime(X,theta,0)
                    ,Yprime(X,theta,0),0),length=len*2,
                        height=len*2, width=.5)
                emitter.rotate(axis=vec(0,1,0),angle=radians(90))
                emitter.rotate(axis=vec(0,0,1),angle=-theta)
                #incoming wave
                c =
                    curve(vec(Xprime(X,theta,Y),Yprime(X,theta,Y
                    ,vec(Xprime(X+deltaX,theta,Y),Yprime(X+deltaX
                #Wave 1
                while(X < len*2):
                    rate(100000)
                    #this loop checks for impacts of
                        all spheres
                    tmp =
                        vec(Xprime(X,theta,Y),Yprime(X,theta,Y),Z
                    for a in atoms:
                        if(mag(a.pos-tmp) < r):
                            atoms.remove(a)
                            newWave(X,vec(tmp.x-Xprime2(X,theta,Y)
                                ,tmp.y-Yprime2(X,theta,Y),0),theta
                    X=X+deltaX
                    c.append(tmp)
                c.visible = False
                del c
                atoms = atomsCopy[:]
        print(b + " theta, " + lam + " lambda: " +
            avgDiff(Yvals))
        emitter.visible = False
        del emitter
```

```
##############Wave Code########################
#X and Y prime are for outgoing wave
def Xprime(X,theta,Y):
    return
        X*cos(-theta)-(Y+cos(((2*pi)/lam)*X))*sin(-theta)

def Yprime(X,theta,Y):
    return
        X*sin(-theta)+(Y+cos(((2*pi)/lam)*X))*cos(-theta)

#X and Y prime2 are for reflected waves
def Xprime2(X,theta,Y):
    return
        X*cos(theta)-(Y+cos(((2*pi)/lam)*X))*sin(theta)

def Yprime2(X,theta,Y):
    return
        X*sin(theta)+(Y+cos(((2*pi)/lam)*X))*cos(theta)
```

```
##############Graphing Code########################
#to be made
```

```
##############Final Simulation########################
GlowScript 3.1 VPython
scene.range = 100
#Crystal Code
#list of avgDiffs of each run
avgDiffs = []
angles = []
#bond-length
bl=3.345
#radius of polonium
r=1.45
#lattice size (#of atoms across)
ls=5
```

```
#length of each side of lattice in angstroms
len=ls*bl
#list to hold all atoms
atoms = []
for X in range(-len/2,len/2,bl):
    for Y in range(-len/2,len/2,bl):
        for Z in range(-len/2,len/2,bl):
            #create sphere and add to list of atoms
            atoms.append(sphere(pos =
                vec(X,Y,Z),radius=r,color=color.blue))
#copy of atom list to use after removal of object
    in reflection code
atomsCopy = atoms[:]

Upper = 24.5
lower = 21
increments = 15
count2 = 0
for b in
    range(lower,Upper,(Upper-lower)/increments):
    #lam = 1.542
    lam = .8
    #list to hold ending y-values
    Yvals = []
    X = -100
    Xinit = X
    deltaX = .01
    theta = radians(b)
    emitter =
        box(pos=vec(Xprime(X,theta,0),Yprime(X,theta,0),0),length=21,
        height=len*2, width=.5)
    emitter.rotate(axis=vec(0,1,0),angle=radians(90))
    emitter.rotate(axis=vec(0,0,1),angle=-theta)
    c = curve()
    c.retain = 2000
    for Z in range(-len+bl,len-bl,3):
        for Y in range(-len+bl,len-bl,3):
            #Reflection Code
            X=-100
            #the starting y value
            initY = Yprime(X,theta,Y)
            #incoming wave
            c.append(vec(Xprime(X,theta,Y),Yprime(X,theta,Y),Z)
                ,vec(Xprime(X+deltaX,theta,Y),Yprime(X+deltaX,theta,Y),Z))
            #Wave 1
            while(X < len*2):
                rate(100000)
                #this loop checks for impacts of all
                    spheres
                tmp =
                    vec(Xprime(X,theta,Y),Yprime(X,theta,Y),Z)
                for a in atoms:
                    if(mag(a.pos-tmp) < r):
                        atoms.remove(a)
                        newWave(X,vec(tmp.x-Xprime2(X,theta,Y)
                            ,tmp.y-Yprime2(X,theta,Y),0),theta,Y)
                X=X+deltaX
                c.append(tmp)
            c.clear()
            atoms = atomsCopy[:]
    avgDiffs.append(1/avgDiff(Yvals))
    Yvals.clear()
    angles.append(b)
    emitter.visible=False
    count2 = count2 + 1
    print(count2)
#graphing code
f1 = gcurve(color=color.green)
dat2 = 10
for i in range(0,11):
    f1.plot(angles[i],avgDiffs[i])

#These functions calculate sinusoidal wave with the
    axes rotated
#wave 1
def Xprime(X,theta,Y):
    return
        X*cos(-theta)-(Y+cos(((2*pi)/lam)*X))*sin(-theta)

def Yprime(X,theta,Y):
    return
        X*sin(-theta)+(Y+cos(((2*pi)/lam)*X))*cos(-theta)

def Xprime2(X,theta,Y):
    return
        X*cos(theta)-(Y+cos(((2*pi)/lam)*X))*sin(theta)

def Yprime2(X,theta,Y):
    return
        X*sin(theta)+(Y+cos(((2*pi)/lam)*X))*cos(theta)

#reflected wave
def newWave(X,diff,theta,Y):
    deltaX=.01
    c2 = curve(vec(Xprime(X,theta,Y)
        ,Yprime(X,theta,Y),Z),vec(Xprime(X+deltaX,theta,Y)
        ,Yprime(X+deltaX,theta,Y),Z))
    c2.retain = 2000
    while(X < -Xinit):
        rate(100000)
        X=X+deltaX
        tmp =
            vec(Xprime2(X,theta,Y),Yprime2(X,theta,Y),Z)
        diff2 =
            vec(Xprime(X,theta,Y)-tmp.x,Yprime(X,theta,Y)-tmp.y
        c2.append(tmp+diff)
    c2.clear()
    Yvals.append(Y+cos(((2*pi)/lam)*X))

#average difference
def avgDiff(list):
    sum = 0
    count = 0
    for i in range(list.length):
        for j in range(i+1,list.length):
            sum=sum+abs(list[i]-list[j])
            count=count+1
    return sum/count

    #
```

B. Simulation Graphs

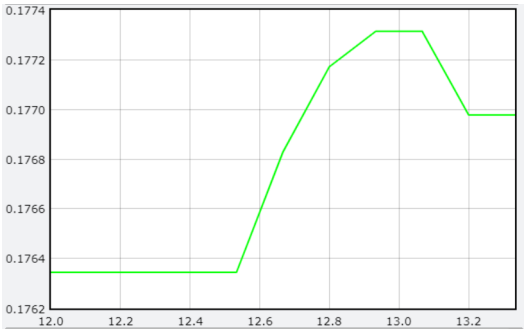

FIG. 6. Simulation Graph from 12-13.5 Degrees (15 iterations)

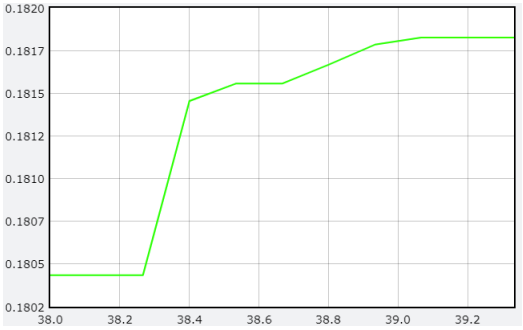

FIG. 7. Simulation Graph from 38-40.5 Degrees (15 iterations)

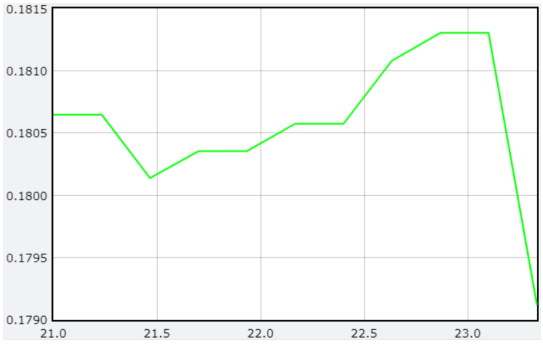

FIG. 8. Simulation Graph from 21-23.5 Degrees (15 iterations)

[1] J. Fiala, "D. l. bish, j. e. post (eds). modern powder diffraction. mineralogical society of america, washington, 1989, xi + 369 p, 167 figures, $ 20.00, isbn 0-939950-24-3," Crystal Research and Technology **25**, 1358–1358 (1990), https://onlinelibrary.wiley.com/doi/pdf/10.1002/crat.2170251126.

[2] W. Wu, M. Balci, S. Song, C. Liu, M. Fokine, F. Laurell, T. Hawkins, J. Ballato, and U. J. Gibson, "Co2 laser annealed sige core optical fibers with radial ge concentration gradients," Opt. Mater. Express **10**, 926–936 (2020).

[3] F. Leitão Muniz, "Simulation of x-ray powder diffraction patterns using dynamical theory," (2015) pp. 55–55.

[4] G. E. Jauncey, eng"The scattering of x-rays and bragg's law," Proceedings of the National Academy of Sciences of the United States of America **10**, 57–60 (1924), 16576781[pmid].

[5] M. Hadjiantonis, *Bragg's law schematics* (2013).

[6] P. J. Schields, "How waves reveal the atomic structure of crystals," (2001).

[7] "An introduction to x-ray powder diffraction analysis,".

[8] C. R. Maxwell, "Physical properties and crystal structure of polonium," Retrospective Theses and Dissertations (1946), 10.31274/rtd-180813-15011.

[9] U. o. S. Mark Winter and W. Ltd, "Polonium: radii of atoms and ions,".

[10] C. Suryanarayana and G. Norton, *X-Ray Diffraction: A Practical Approach* (Springer Science Business Media, 2013).