A VISUALIZATION TOOL FOR PHOTON POLARIZATION TOMOGRAPHY

David Wacaser[1], Patrick Snyder[1], and Ali Passian[2]

[1]Department of Physics, University of Illinois Urbana-Champaign, Urbana, IL 61801-3003, USA

[2]Quantum Information Science Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, USA

Summer 2022

ABSTRACT

Emerging applications of quantum information science (QIS) such as quantum sensing, quantum computing, and quantum communications promise powerful new capabilities that have the potential to revolutionize science and technology in the coming years. Common to many of these nascent applications is the use of the particles of light, the photons. The properties of light, such as polarization of a photon, can be used to represent information. Classically, information is represented with a digital bit. In QIS, the counterpart of a bit is called a qubit. Since working with the polarization states of photons or qubits enable information processing, methods to study the polarization are needed. Polarization tomography allows the statistical determination of the polarization states of photons. Here, we present a graphical approach to visualization and animation of the polarization state of photons. Using C# as the programming platform, the software package Unity is used to simulate and animate the movement of photons through logic gates that form the basis for quantum computing. The polarization is visualized on a Bloch sphere. The developed program can be made available upon request.

I. Background and Introduction

Light has a unique property called polarization. The polarization can either be vertical or horizontal. If you isolate a single photon, or packet of light, it will have a certain polarization. However, when you have only a single photon besides only being able to be horizontal or vertical it can also be a superposition of both horizontal and vertical at the same time. This leaves us with a probability of it being horizontal or vertical when measured. This allows a photon to be used as a quantum bit (qubit). Because it is able to be both horizontally and vertically polarized at the same time it is sometimes hard to visualize. For this reason an animation or simulation of a polarized photon can help someone visualize the polarization of a photon.

The best way to visualize the polarization of a photon is using the Bloch Sphere.[1] It was first suggested at the end of the 19th century by Henri Poincaré.[2] The Bloch Sphere uses spherical coordinates in order to describe the superposition of two states in a way that is both less computationally heavy and easier to visualize.

One of the easiest ways to change the state of the Bloch Sphere is by applying a gate to it. A photon's polarization state changes all the time from various factors, so a gate is a great way to simulate that. Gates can be written as a matrix and when multiplied by the state vector they change the state of the sphere in some predetermined way. There are many filters that light goes through that can be modeled like a gate in order to change the state of the photon. Some filters and gates along with their matrices include:

- X Gate - $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
- Y Gate - $\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
- Z Gate - $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
- Hadamard Gate - $\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$
- Horizontal Linear Polarizer - $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
- Vertical Linear Polarizer - $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
- Linear Polarizer $45°$ - $\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$
- Linear Polarizer -$45°$ - $\begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}$
- Horizontal Quarter Waveplate - $\begin{bmatrix} \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i & 0 \\ 0 & -\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i \end{bmatrix}$
- Vertical Quarter Waveplate - $\begin{bmatrix} \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i & 0 \\ 0 & \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}i \end{bmatrix}$
- Right Circular Polarizer - $\begin{bmatrix} 0.5 & \frac{i}{2} \\ -\frac{i}{2} & 0.5 \end{bmatrix}$
- Left Circular Polarizer - $\begin{bmatrix} 0.5 & -\frac{i}{2} \\ \frac{i}{2} & 0.5 \end{bmatrix}$

Our goal in this project is to make a program that will help demonstrate the polarization of a photon and what happens as it goes through various filters. And eventually to determine the polarization of an unknown polarization state by running it through various filters as well.

II. Method

In order to animate the polarization of a photon and also to allow the user to modify the way in which the photon goes through gates and filters, Unity - a game development software - was deemed the best tool for the job.

II.I Unity

As mentioned above Unity is a game development software. Using a game development software provided a few advantages. It allowed us to easily build a Graphical User Interface (GUI) with buttons and text. It also provided us with some building blocks to make the program work visually. We created a sphere object to represent the block sphere, and used very flat cylinders to create the axis of the sphere. We then attached a cylinder within the sphere to represent the Bloch Vector.

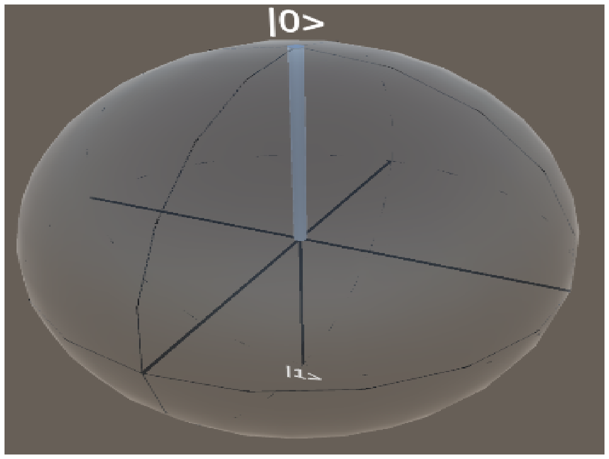


Figure 1: The Bloch Sphere created in Unity

One of the great features of Unity was that it made it easy to modify the visual aspects while also controlling some aspects with scripts. Using a C# script we can rotate the vector within the sphere, and also move the sphere itself keeping everything inside of it. Besides moving and rotating the Bloch Sphere we also used scripts to switch between cameras, add filters, and many other functions.

II.II Complex Numbers

Unity by default uses C# scripts, and while it is possible to make it accept other languages it is significantly easier to use C#. One of the issues with C# is that the latest version has no support for imaginary or complex numbers. Because of this, we had to create a custom library in order to implement them. We created a new object to represent complex numbers which included two parts, a real and an imaginary part. We also added some operations that can be done on or by complex numbers such as addition, subtraction, and multiplication. Because some of the matrices that describe the gates include imaginary numbers we had to also make a new matrix object that used complex numbers.

II.III Rotating The Vector

After changing the state of the photon we need to rotate the vector in the Bloch Sphere. The easiest way to rotate it is using spherical coordinates, so we need to be able to convert between a state vector $\begin{bmatrix}\alpha \\ \beta\end{bmatrix}$ and the angles ϕ and θ. We are able to find ϕ by the equation $\phi = ATan2(\beta_{imaginary}, \beta_{real})$ and we can find θ with the equation $\theta = 2cos^{-1}(\alpha)$.

II.IV Adding Filters

Now that we can rotate the Bloch Vector by changing it's state we need ways to change the state. We added various filters that the photon can be moved through which will alter it's state. We then added a GUI to allow for adding and removing filters during run time. The GUI allows you to select which filter to add, and it's position. And for the Linear Polarizer it allows you to select the rotation as well.

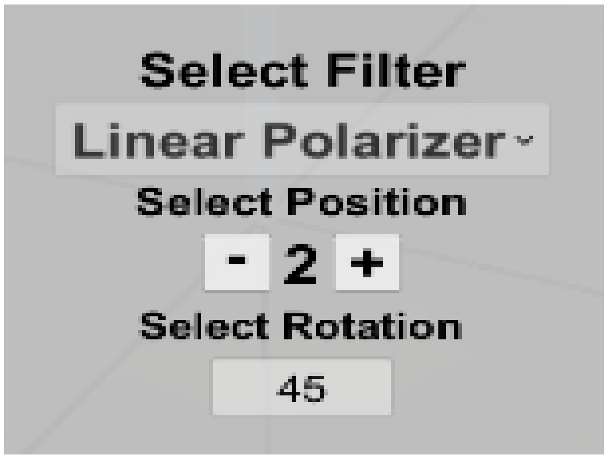
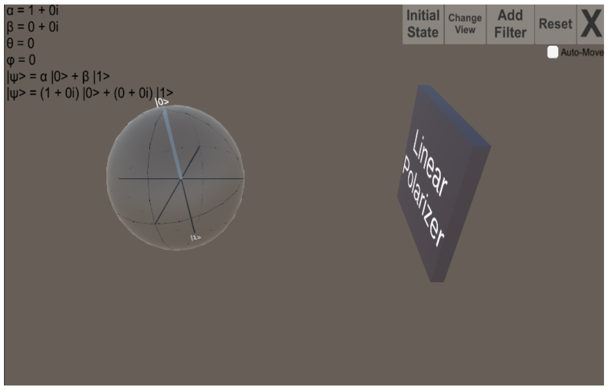


Figure 2: The Add Filter GUI

If you want to remove a filter you can select 'None' from the list of Filters and select the position of the filter you wish to remove.

III. Results

Over the course of this summer we designed a program to simulate the polarization of a photon, and then we started creating it using Unity. The program lets you add various filters at various positions. It also allows you to move a photon through the filters at your own pace and watch the state change as it goes through them. It shows the state, as well as the angles in spherical coordinates and will even let you adjust the initial state of the photon. We put the program on GitHub and you can find it by following the link in the Appendix.



$\alpha = 1 + 0i$
$\beta = 0 + 0i$
$\theta = 0$
$\varphi = 0$
$|\psi> = \alpha\ |0> + \beta\ |1>$
$|\psi> = (1 + 0i)\ |0> + (0 + 0i)\ |1>$

Figure 3: A screenshot of the program

IV. Conclusions

This program was mainly designed to help visualize the polarization of a photon. It could be useful for someone learning about polarization or even for someone doing research on polarization. There is still more work that can be done on it that we did not have time to do over the course of this summer. Further work could include adding a rotation parameter to the waveplates, a way to input a photon in an unknown polarization state in order to determine the most likely polarization state of it, and outputting the result into a Maximum Likelihood Estimator.

V. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant PHY-1950744. This site is supported by the Department of Defense in partnership with the NSF REU program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the Department of Defense.

VI. References

1. Glendinning, Ian. ”The bloch sphere.” In QIA Meeting. 2005.

2. Poincaré, Henri. Théorie mathématique de la lumière II.: Nouvelles études sur la diffraction.–Théorie de la dispersion de Helmholtz. Leçons professées pendant le premier semestre 1891-1892. Vol. 1. G. Carré, 1889.

3. Parakh, Abhishek, Mahadevan Subramaniam, and Elliott Ostler. ”Quasim: A virtual quantum cryptography educator.” In 2017 IEEE International Conference on Electro Information Technology (EIT), pp. 600-605. IEEE, 2017.

4. Toninelli, Ermes, Bienvenu Ndagano, Adam Vallés, Bereneice Sephton, Isaac Nape, Antonio Ambrosio, Federico Capasso, Miles J. Padgett, and Andrew Forbes. ”Concepts in quantum state tomography and classical implementation with intense light: a tutorial.” Advances in Optics and Photonics 11, no. 1 (2019): 67-134.

VII. Appendix

GitHub link for .exe file:

`https://github.com/captainjolt/PhotonPolarization/blob/main/PhotonPolarization.exe`