3D Computer Modeling Electron Orbitals

Lorelli, Stephanie
Siu, Brandon*
San Jose State University

(Collaboration with Professor Romanowsky)
(Dated: December 9, 2015)

3D computer modeling has unlocked the potential for visualizing systems that are too small to see with our eyes. The atom is the building block of life, but we cannot see an atom with our eyes nor can we see individual electrons orbiting the nucleus of the atom. Therefore, many people have a difficult time imagining how this system works. Software has been created that can trace pathways of orbitals and animate particles following those paths. This can be accomplished by using packages called Matplotlib and Mayavi. Also viewing electron orbital shapes can be made interactive with a widget in IPython Notebook so that the viewer can switch between the different shapes of the orbitals. We have accomplished these tasks through the use of a particular programming language called Python. Once these animations were made, they were all consolidated into one website. We hope to make visualizing atoms and electron orbitals easier and more accessible for everyone.

I. INTRODUCTION

For many people it is hard to visualize moving systems in 3D. This is especially difficult for tiny systems such as atoms and electrons that are invisible to the naked eye and move at the speed of light. Hence, we usually only see still 2D photographs in books such as Figure 1 from *Where Are We?* by Christopher Madden [2]. However, this picture is not even remotely accurate as to how electrons randomly move within the electron levels of the atom. The picture is over simplified because random patterns within different shells at different radii are extremely hard to show in a 2D picture. We hope to make a more accurate representation of how electrons actually behave within the atom. Vogt et al. (2015)[3] hopes to promote a change for accessing 3D plots by using X3D pathway. This will allow 3D plots to be available to more people who may not have programming software via web pages.

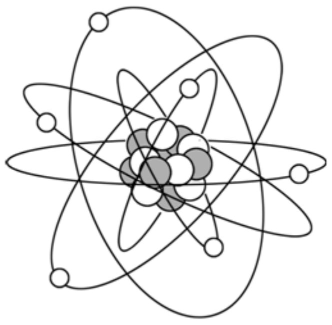


FIG. 1. A basic 2D visualization of an atom commonly found in textbooks[2].

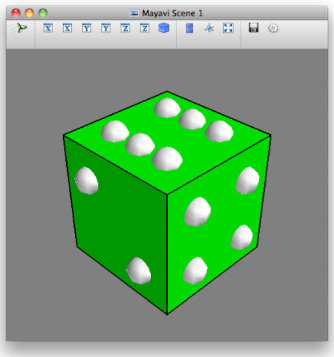


FIG. 2. A 3D image of dice as produced in Mayavi [3]

The first goal is to make a non-animated 3D model of the atom's orbitals from classical mechanics in both Mayavi and Matplotlib. From there, the next goal will be to animate electrons following the paths around the nucleus. Once these animations are working, we will try to reproduce different orbital shapes in Matplotlib and combine them in an interactive widget for people to experiment with.

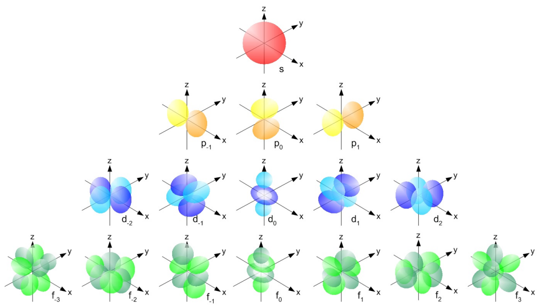


FIG. 3. Orbital Shapes of Electrons [1]

Finally, we will put all of our plots together so that

* Physics Department, San Jose State University.

more people can access them in one place; such as, a website.

II. METHODS

First, Siu plotted one orbit in Mayavi around a fixed sphere. Lorelli also did the same but in Matplotlib. More orbits were then added at different radii to model a real atom in classical mechanics. Once this was figured out, then animating electrons traveling the orbits was the next step. The animations would help to make the motions of the electrons appear more chaotic.

Lorelli first tried to make a simple plot of beryllium in Matplotlib. Initially she was able to get a circle plotted using spherical coordinates shown in the following equations:

$$x = rsin(\theta)cos(\phi) \tag{1}$$

$$y = rsin(\theta)sin(\phi) \tag{2}$$

$$z = rcos(\theta) \tag{3}$$

However, trying to get the nucleus aligned with the center of the atom was a bit tricky. Whenever the ϕ or θ angles were adjusted, the whole circle moved around in the plot. As soon as one circle was aligned, the next circle would be misaligned. She finally got all the circles to align after adjusting the values of ϕ and θ multiple times within the orbital equations and adjusting and where the nucleus was plotted. Then Lorelli made multiple orbits to represent the 4 electrons that beryllium has. Two were closer to the nucleus and two were farther from the nucleus. This was to represent the electron levels where atoms can have up to two electrons in the first shell and then the next shell can have up to 8 electrons. She then adjusted the ϕ angle so that each orbital would surround the nucleus from all sides at any angle.

After this step, Lorelli figured out how to animate electrons around a nucleus and keep the animation running infinitely. This was done by attaching a time array to the end of each θ instead of having θ go from 0 to 2π. A snapshot of this plot is shown in Figure 4. However, this classical mechanics representation is inaccurate as to how electrons actually behave within an atom. Electrons have the ability to move randomly within electron clouds. This is a completely different picture than electrons always following one path around the nucleus. To make the 3D plot more accurate, Lorelli rotated the orbits to surround the entire nucleus like a sphere by adding the time vector to both the ϕ and θ angles and then she sped up the electrons' velocities to appear to move randomly within each electron cloud shell as in Figure 5. This represents how an s orbital should look like.

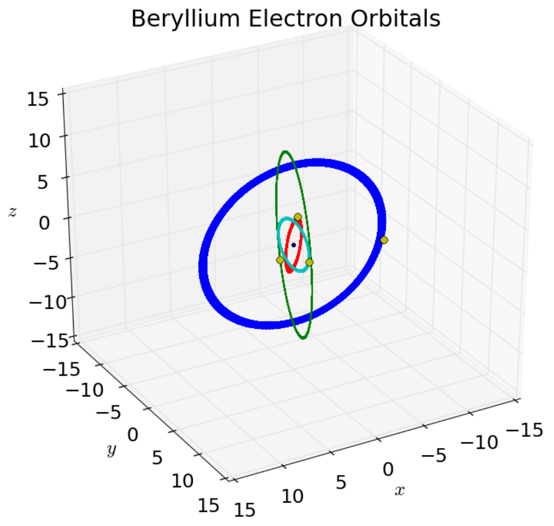


FIG. 4. Picture of the animated electrons circling the nucleus infinitely

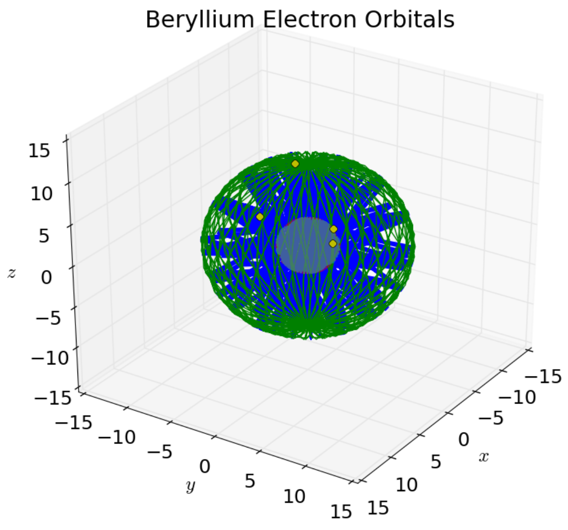


FIG. 5. Picture of the animated electrons randomly orbiting the nucleus in spherical shells

A. Python code Orbital

This code was created by Stephanie Lorelli to generate a plot of the electron orbitals in an atom as shown in Figure 4.

```
# phys 40 electrons around an atom
```

```
from pylab import *
from matplotlib.animation import *
from mpl_toolkits.mplot3d import Axes3D

#runs indefinitely

theta_max = 360.0
theta_rad = radians(theta_max)

# time vector :
t = linspace(0.,10.0*2.0*pi,1000)


#orbital 1
n_degrees = 7.0*360.0
n_degrees2 = 5.0*360.0


r1 = 10.0
theta1 = 2.0*theta_rad * (t)
phi1 = radians(180.0)


x1 = r1*sin(theta1)*cos(phi1)
y1 = r1*sin(theta1)*sin(phi1)
z1 = r1*cos(theta1)

#orbital 2
r2 = 10.0
theta2 = theta_rad * (t)
phi2 = radians(70.0)


x2 = r2*sin(theta2)*cos(phi2)
y2 = r2*sin(theta2)*sin(phi2)
z2 = r2*cos(theta2)

#orbital 3
r3 = 3.0
theta3 = 3.0*theta_rad * (t)
phi3 = radians(45.0)


x3 = r3*sin(theta3)*cos(phi3)
y3 = r3*sin(theta3)*sin(phi3)
z3 = r3*cos(theta3)

#orbital 4
r4 = 3.0
theta4 = 2.0*theta_rad * (t)
phi4 = radians(90.0)


x4 = r4*sin(theta4)*cos(phi4)
y4 = r4*sin(theta4)*sin(phi4)
z4 = r4*cos(theta4)
#nucleus
x_n = 0.0
y_n = 0.0
z_n = 10.0

rcParams.update({'font.size': 18})

# set up figure frame:
fig1 = figure(figsize=(10,10))
ax = fig1.gca(projection='3d')

plot(x1,y1,z1)
plot(x2,y2,z2)
plot(x3,y3,z3)
plot(x4,y4,z4)
scatter(x_n,y_n,z_n)


electron1, = ax.plot([],[],[],
'oy',markersize=7)
electron2, = ax.plot([],[],[],
'oy',markersize=7)
electron3, = ax.plot([],[],[],
'oy',markersize=7)
electron4, = ax.plot([],[],[],
'oy',markersize=7)

xlabel('$x$')
ylabel('$y$')
ax.set_zlabel('$z$')
ax.set_xlim(-15.0,15.0)
ax.set_ylim(-15.0,15.0)
ax.set_zlim(-15.0,15.0)
ax.view_init(24,-136)
title('Beryllium Electron Orbitals')



def trajectory_init():
# initialize blank frames
    electron1.set_data([],[])
    electron1.set_3d_properties([])
    electron2.set_data([],[])
    electron2.set_3d_properties([])
    electron3.set_data([],[])
    electron3.set_3d_properties([])
    electron4.set_data([],[])
    electron4.set_3d_properties([])
    return electron1,electron2, electron3,
    electron4

def trajectory_frame(i):
    x_1 = x1[i]
    y_1 = y1[i]
    z_1 = z1[i]
    electron1.set_data(x_1, y_1)
    electron1.set_3d_properties(z_1)
    x_2 = x2[i]
    y_2 = y2[i]
    z_2 = z2[i]
    electron2.set_data(x_2, y_2)
    electron2.set_3d_properties(z_2)
    x_3 = x3[i]
    y_3 = y3[i]
    z_3 = z3[i]
    electron3.set_data(x_3, y_3)
    electron3.set_3d_properties(z_3)
    x_4 = x4[i]
    y_4 = y4[i]
    z_4 = z4[i]
    electron4.set_data(x_4, y_4)
    electron4.set_3d_properties(z_4)
    ax.view_init(30,0.3*i)
    fig1.canvas.draw()
    return electron1,electron2,
    electron3,electron4

# 50 ms delay:
ani = FuncAnimation(fig1,
trajectory_frame,
init_func=trajectory_init, frames=400,
interval=50, blit=True)


show()
```

Once both electron orbitals were created, then we wanted to create a widget. Lorelli figured out how to separately draw and populate s and p orbitals with slightly transparent points as shown in Figures 7 and 8. She also figured out how to exclude points that were out of bounds of the circle by finding them with a ”for loop”, asking an ”if and elif” statement, and replacing those numbers with ”None” so that those points would not be plotted. Siu made d orbitals that were rotated at an angle θ as shown in Figure 9. Afterwards we incorporated these plots into a single widget in Matplotlib. When this was successful, the last step was to combine all the plots, movies, and gif animations into one website. This way more people would have access to viewing the electron orbital systems in 3D. No coding software will be required to view the plots, however IPython Notebook will be required to view the widget. Images from the widget are shown in Figures 7,8, and 9.

B. Python code Widget

This code was created by Stephanie Lorelli to exclude points in Figure 7.

```
#exclude points in circle
    for i in range(0,len(x2)):
        if x2[i]>18.0:
            x2[i] = None
        elif x2[i]<-18.0:
            x2[i] = None
        elif y2[i]<-15.0:
            y2[i] = None
        elif y2[i]>18.0:
            y2[i] = None
        elif x4[i]> 18.0:
            x4[i] = None
        elif x4[i]<-18.0:
            x4[i] = None
        elif y4[i]<-18.0:
            y4[i] = None
        elif y4[i]>18.0:
            y4[i] = None

        x2_2 = x2
        y2_2 = y2
        x4_2 = x4
        y4_2 = y4
```

III. RESULTS AND DISCUSSION

The results for this project were that Siu was able to produce electrons orbiting a nucleus in Mayavi as shown in Figure 6. Lorelli was able to produce animated plots of electrons orbiting a nucleus as shown in Figures 4 and 5. This code took many trial, error, and debugging techniques to finally get it to produce the correct plot that was desired.

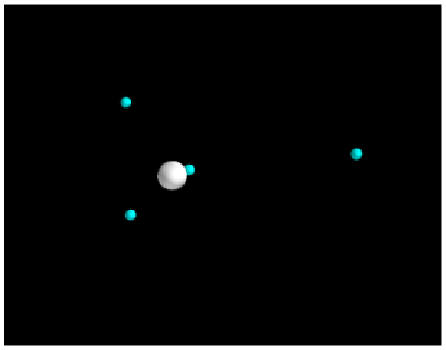


FIG. 6. A 3D image electrons orbiting a nucleus produced in Mayavi by Siu.

However to fully appreciate all the work at a glance, Lorelli created a website as shown in Figures 10 and 11 using wix.com [4] to put all of the animations together. The website address is http://slorelli01.wix.com/electron-orbitals3d. After many hours of struggling, the results were able to accomplish our goals of visualizing how electrons move within an atom in 3D. We also were able to switch between different orbital shapes rapidly with a widget. We also accomplished the goal of making the animations and plots easily accessible to everyone via a website. Included in

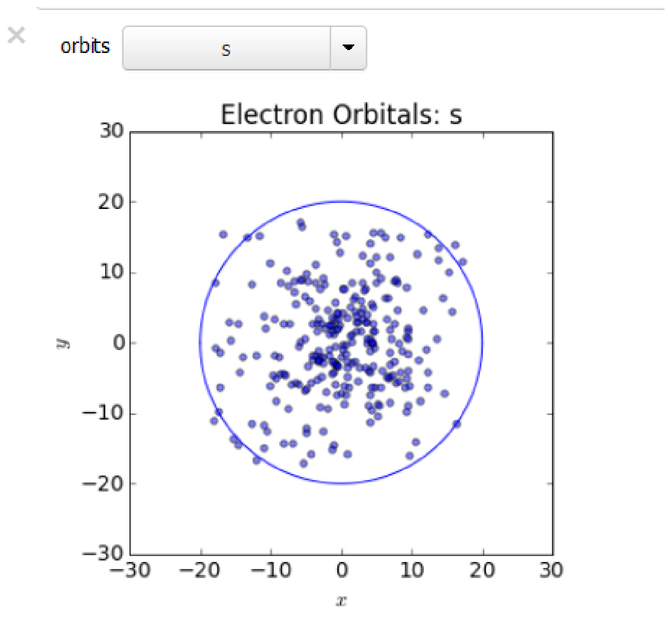
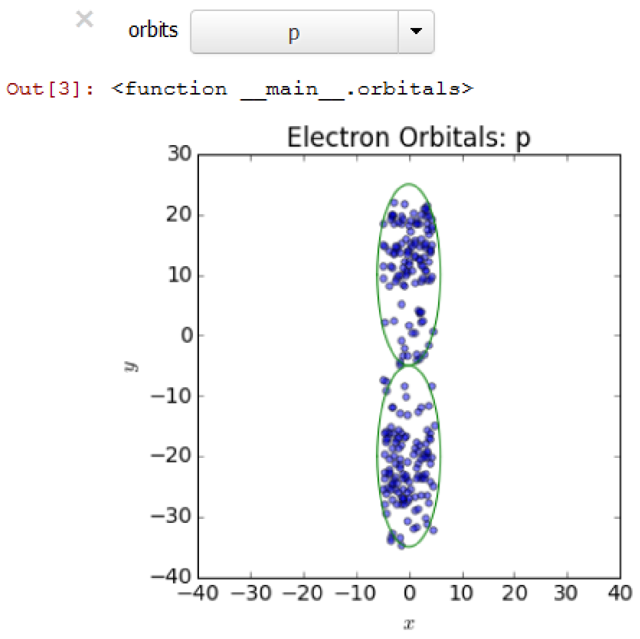


FIG. 7. S Orbital Plot in widget



```
Out[3]: <function __main__.orbitals>
```

FIG. 8. P Orbital Plot in widget

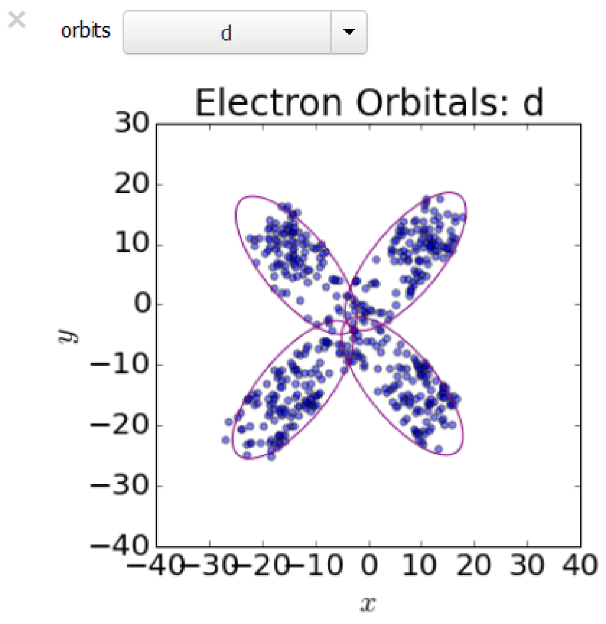


FIG. 9. D Orbital Plot in widget

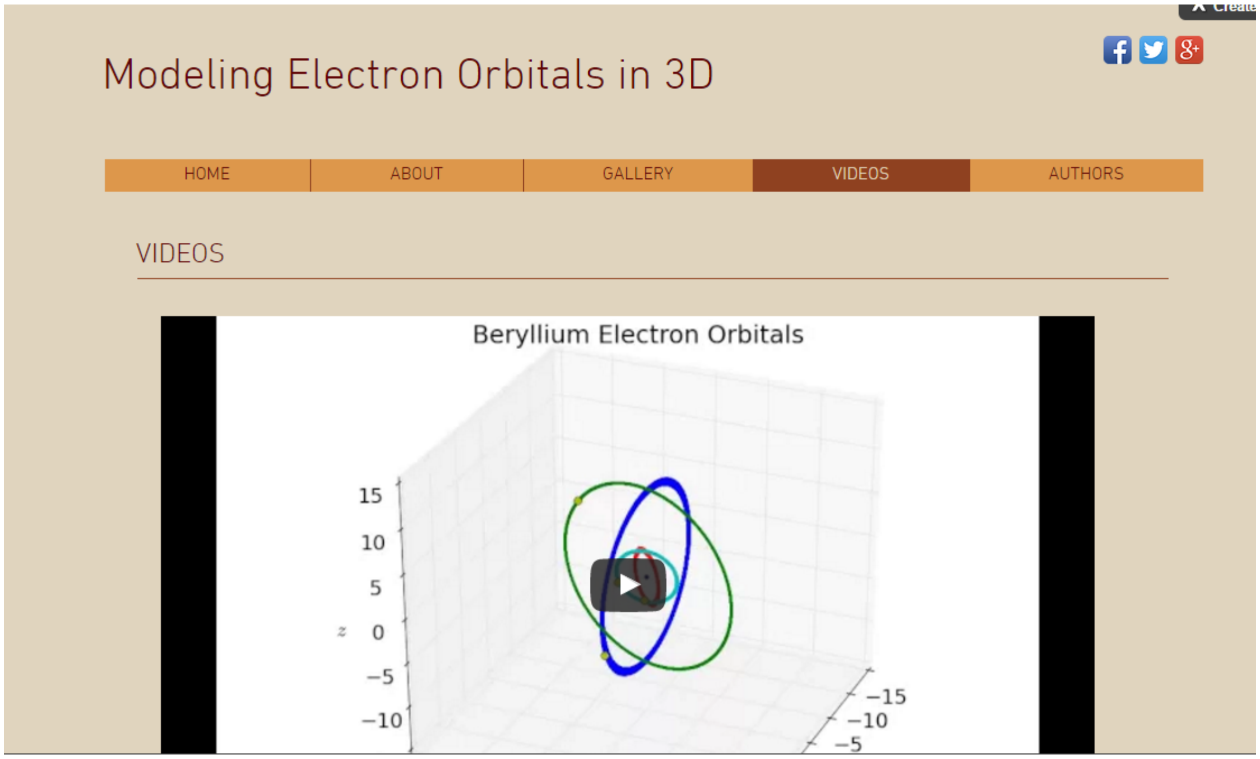
the website are figures that were saved from the plots in .mp4 format and .gif format.

The gif was generated by saving .png pictures of the plot at every interval of time. There was a slight problem when generating the png images because namespaces were left out of the original code. Python did not know whether to use Pylab, TVTK Viewer, or Mayavi. We had to fix the namespaces to specify where each item was being called from and have the plot show up in Mayavi instead of the TVTK Viewer. These png images were then converted via a ffmpeg converter to .gif format in the command line of Windows. The converter put all of the pictures together and runs through them at a rate to create the gif file. X3D [3] was also utilized in the website after the X3D code was converted to HTML code via *3D encoding converter* [5]. This code was then pasted into an add on for the website. The X3D file allows people to rotate the 3D plot without installing coding software.

Modeling Electron Orbitals in 3D

Coding is speaking the language of computers.

FIG. 10. Home page of Website



Modeling Electron Orbitals in 3D

FIG. 11. Videos page of Website

IV. CONCLUSION

By generating 3D plots in different packages such as Mayavi and Matplotlib we were able to learn more about the program and how it operates. Mayavi updates every particle's position as it runs through the animation while Matplotlib runs through the scenes of the plot that has been previously calculated. These are two different approaches to accomplishing the goal of animating systems in 3D.

By having the computer make visible what would normally be invisible, we were able to see atoms come alive in 3D animations. For future coders, the question now is, "What else is possible to accomplish in 3D models?" There is no limit to the imagination. Suggestions for people in the future wanting to generate 3D animations would be to do much research, practice coding, and ask questions. These will be the keys to debugging effectively and making the project happen. The more one gets familiar with the tools of coding, the more it is possible to accomplish. It becomes easier to see how coding can be applied in daily life to model different equations or data sets. Coding also opens up opportunities. Within 4 months someone can go from having zero coding experience to creating websites and 3D animations. This is an incredible feat of the power of learning a coding language of computers. Finally, it was rewarding to put all of the hard work together in one cohesive website at http://slorelli01.wix.com/electron-orbitals3d that many people can view easily.

ACKNOWLEDGMENTS

We wish to acknowledge the support of the classmates, instructors, and Google for helping with this project.

[1] *Electronic Orbitals.* UC Davis ChemWiki. n.d. Web. December 6, 2015.
[2] Madden, Christopher. *Where Are We?*, Inkline Press, n.d. Web. November 28, 2015.
[3] Vogt et al. *Advanced data visualization in astrophysics: the X3D pathway*, (2015)
[4] *Wix*, Wix.com, Inc. 2006. Web. November 28, 2015.
[5] *X3D encoding converter*, instantlabs. n.d. Web. Dec.6, 2015.