VIRTUAL LABORATORY OF THIN LENSES USING ANDROID SMARTPHONE WITH PYTHON PROGRAMMING

Catur Edi Widodo
Department of Physics,
Diponegoro University, Semarang, Indonesia
Catur.ediwidodo@gmail.com

Publication History
Manuscript Reference No: IJIRAE/RS/Vol.09/Issue05/MYAE10086
Research Article | Open Access
Peer-review: Double-blind Peer-reviewed
Article ID: IJIRAE/RS/Vol.09/Issue05/MYAE10086
Received Date: 12, May 2022 | Accepted Date: 25, May 2022 | Available Online: 31, May 2022
Volume 2022 | Article ID MYAE10083 http://www.ijirae.com/volumes/Vol9/iss-05/05.MYAE10086.pdf
Article Citation: Catur (2022). Virtual Laboratory of thin Lenses using Android Smartphone with Python Programming, International Journal of Innovative Research in Advanced Engineering (Vol. 9, Issue 5, pp. 128–132). AM Publications, India **doi:** https://doi.org/10.26562/ijirae.2022.v0905.05
BibTeX key
Academic Editor-Chief: Dr.A.Arul Lawrence Selvakumar

Abstract: A computer program has been created to simulate a ray diagram on a thin lens using the Python programming language. The program is a ray diagram simulation on a convex lens and a concave lens. The program is built using the canvas, create line, button and motion functions in the object-oriented programming language Python. The object being simulated is a convex and concave lens with a focal length of 10 cm and an object's height of 4 cm. The simulation results are in the form of an interface with the user using computer graphic media and interaction via a touch screen. The program can run on Android mobile phones. The results of ray diagrams on computers and Android cellular phones show that they are in accordance with theory, so that the program can be used as a virtual laboratory material.
Keywords: virtual laboratory, computer program, simulation, lenses

I. INTRODUCTION

Computer simulations are made to make it easier for humans to study, observe, and predict physical phenomena that may occur. In principle, simulation can be done in various ways, for example with a series of numbers, pictures, graphics, or visualization with a computer. With computer-based simulations, the costs incurred can be minimized because the modeling does not have to be in the real world with the actual size. With computer-based simulation, the risk of research accidents can be reduced to zero percent [1]. In certain circumstances, simulation can replace laboratory experiments. Some analogies between laboratory experiments and computer simulations can be seen in table 1.

Table 1. Analogy of experiments and simulations [2]

Laboratory experiments	Computer simulation
Sample	Model
Equipment	Program
calibration	Testing
Measurement	Computing
Analysis	Analysis

Computer simulations, as well as laboratory experiments, are tools that can be used to study complex physical phenomena. Computer simulations can be used to complement laboratory experiments as a means of obtaining knowledge in a practical and effective manner [3]. A virtual laboratory is a computer-based activity in which students interact with experimental equipment or other activities via a computer interface [4-5]. Common examples are simulation experiments, in which a student interacts with programmed behavior, and remotely controlled experiments in which a student interacts with real equipment via a computer link. A virtual laboratory is a laboratory where students interact with experiments that do not have direct physical reality. The whole concept of interactive screen experiments is to bring as close to reality as possible, to as many students as possible. Currently, with the large number of cell phones, it is very possible to create virtual laboratory applications that can be accessed using cell phones.

Users can interact with experimental objects via the mobile phone screen by "draging" using their fingers or other tools [6]. The use of this phone can be done online or offline where the cell phone functions the same as a computer. The purpose of this study is to create an application on an Android-based cellular phone to simulate the tracing of light rays on thin convex and concave lenses. With this simulation, the limitations of optical laboratory equipment can be overcome, so that experiments can run more effectively and can be carried out anywhere.

II. THEORY

The most important simple optical instrument is a thin lens. Thin lenses are usually circular in shape and both surfaces are curved. Both surfaces can be convex, concave or flat. The axis of the lens is a straight line that passes through the center of the lens and is perpendicular to its surface. If parallel rays parallel to the axis fall on a thin convex lens, they will be focused at a focal point, F. Rays of light that from a point on a distant object are essentially parallel, so it can be said that: the focal point is the image point for object at an infinite distance from the principal axis. This means that the focal point of the lens can be found by determining the point at which the rays of sunlight are formed into sharp images. The distance from the focal point to the center is called the focal distance, f. [7].

To find the image point, we can use a two-point beam of light with the following rules for a thin convex lens:

1. A ray parallel to the principal axis is refracted through the focal point
2. The ray that passes through the focus is refracted parallel to the principal axis
3. A beam that passes through the center is refracted without being bent.

The rules for concave lenses are almost the same, namely:

1. A beam parallel to the principal axis is refracted as if from the focal point
2. The beam leading to the focus is refracted parallel to the principal axis
3. A beam that passes through the center is refracted without being bent

By using these rules, the formula for the relationship between object distance (so), image distance (si), focal distance (f), object height (ho), and image height (hi) is obtained as follows:

$$1/ho + 1/hi = 1/f \quad (1)$$

and

$$ho/hi = so/si \quad (2)$$

III. METHOD

The research stages follow the waterfall paradigm software development path which includes requirements analysis, design, implementation, and testing. Stages of requirements analysis includes determining software functions. As input in the analysis is a description of the system to be built. The system that will be built consists of a convex lens system and a concave lens system with upright objects that can be shifted with the mouse away from or near the lens. Shadow distance and size follow formulas (1) and (2). The design stage includes the design of system architecture, interfaces and algorithms that are ready to be implemented. The program is designed with object orientation which includes events, objects, classes and user interfaces by using Tkinter in Python.

The implementation stage is in the form of computer program based on the algorithm that has been generated by the design stage. The computer program is implemented in the Python programming language using the canvas, create line, button and motion functions.[7-10]. The testing stage is carried out to find out whether the program is correct. Testing is in the form of verifying whether the software produced is in accordance with the expected specifications and validation whether the functions formed are correct/valid. After the program can function properly, then the program is copied into the cell phone, and the Pydroid software is also installed on the cell phone so that the program can run on the cell phone.Here we present the program code for positive and negative lenses. The first program is the positive lens and the second program is the negative lens.

```
#python code of positive lens
        from tkinter import *
        root = Tk()
        canv = Canvas(root, width=800, height=500)
        root.title ('positive lens')
        xlensa=400
        canv.create_oval(xlensa-5,250, xlensa+5,350,width=0, fill='yellow')
        canv.create_line(0,300,800,300,width=0, fill='blue')
        fokus=100
        so=200
        xso = xlensa-so
        ho = 40
        a=xso
        b=300-ho
        si =0
```

```
hi =0
xsi=0
xhi=0
ai=0
hi=0
canv.create_line(a,300, a ,300-ho,width=4, fill='brown')
def klik(event):
 global a,b,xlensa,so,xso,ho,xho,si,xsi,ai,hi,xai,xhi,bi,fokus
 canv.delete(ALL)
 a=event.x
 b=event.y
 canv.create_line(a,300, a ,300-ho,width=4, fill='brown')
 canv.create_oval(xlensa-5,250, xlensa+5,350, width=0,fill='yellow')
 canv.create_line(0,300,800,300,width=0, fill='blue')
 canv.create_line(400-fokus,300, 400-fokus,305,width=5, fill='green')
 canv.create_line(400+fokus,300, 400+fokus,305,width=5, fill='green')
 canv.create_line(a,300-ho, 400,300-ho, fill='dark red')
 canv.create_line(400,300-ho, 400+si,300+hi, fill='dark red')
 canv.create_line(a,300-ho, 400+si,300+hi, fill='dark red')
 canv.create_line(a,300-ho, 400,300, fill='dark red')
 canv.create_line(400,300-ho, 400+fokus,300, fill='dark red')
 if so < fokus : canv.create_line(400,300, 400-si,300-hi, fill='dark red')
 if so < fokus : canv.create_line(400+fokus,300, 400+fokus+0.5*(fokus-si),300-0.5*hi, fill='dark red')
 xso=a
 so=xlensa-xso
 si= 1.00 /( 1.00/fokus - 1.00/so)
 ai=xlensa + si
 hi= si * ho / so
 bi=300
 canv.create_line(ai,300, ai ,300+hi,width=4, fill='orange')
 print ('so = ',so, ' si = ',si,' ho = ',ho, ' hi = ', hi)
canv.bind('<B1-Motion>', klik)
canv.pack()
root.mainloop()
```

```
#python code of negative lens
from Tkinter import *
root = Tk()
canv = Canvas(root, width=800, height=500)
root.title ('negative lens')
xlensa=400
canv.create_line(xlensa-5,250,xlensa,300,xlensa-5,350,width=5, fill='yellow',smooth=1)
canv.create_line(xlensa,250, xlensa,350,width=5, fill='yellow',smooth=1)
canv.create_line(xlensa+5,250, xlensa,300,xlensa+5,350,width=5, fill='yellow',smooth=1)
canv.create_line(0,300,800,300,width=0, fill='blue')
fokus=100
so=200
xso = xlensa-so
ho = 40
a=xso
b=300-ho
si =0
hi =0
xsi=0
xhi=0
ai=0
hi=0
canv.create_line(a,300, a ,300-ho,width=4, fill='brown')
def klik(event):
global a,b,xlensa,so,xso,ho,xho,si,xsi,ai,hi,xai,xhi,bi,fokus
   canv.delete(ALL)
   a=event.x
```

```
b=event.y
canv.create_line(a,300, a ,300-ho,width=4, fill='brown')
canv.create_line(xlensa-5,250, xlensa,300,xlensa-5,350,width=5, fill='yellow',smooth=1)
canv.create_line(xlensa,250, xlensa,350,width=5, fill='yellow',smooth=1)
canv.create_line(xlensa+5,250, xlensa,300,xlensa+5,350,width=5, fill='yellow',smooth=1)
canv.create_line(0,300,800,300,width=0, fill='blue')
canv.create_line(400-fokus,300, 400-fokus,305,width=5, fill='green')
canv.create_line(400+fokus,300, 400+fokus,305,width=5, fill='green')
canv.create_line(a,300-ho, 400,300-ho, fill='red')
canv.create_line(400,300-ho, 400+si,300+hi, fill='red')
canv.create_line(a,300-ho, 400+si,300+hi, fill='red')
canv.create_line(a,300-ho, 400,300, fill='red')
canv.create_line(0.5*so+400,300+0.5*ho, 400,300, fill='red')
canv.create_line(400,300-ho, 400-fokus,300, fill='orange')
canv.create_line(400,300-ho, 400+fokus,300-2*ho, fill='red')
xso=a
so=xlensa-xso
si= 1.00 /( -1.00/fokus - 1.00/so)
ai=xlensa + si
hi= si * ho / so
bi=300
canv.create_line(ai,300, ai ,300+hi,width=4, fill='orange')
print ('so = ',so, '    si =  ',si)
print ('ho = ',ho, '    hi =  ',hi)
canv.bind('<B1-Motion>', klik)
canv.pack()
root.mainloop()
```

IV. RESULTS AND DISCUSSION

After going through the stages of requirements to testing, the results of ray diagram simulations for convex and concave lenses are obtained as shown in Figure 1 and Figure 2. Meanwhile, the use of the simulation in a cellular phone can be seen in Figure 3

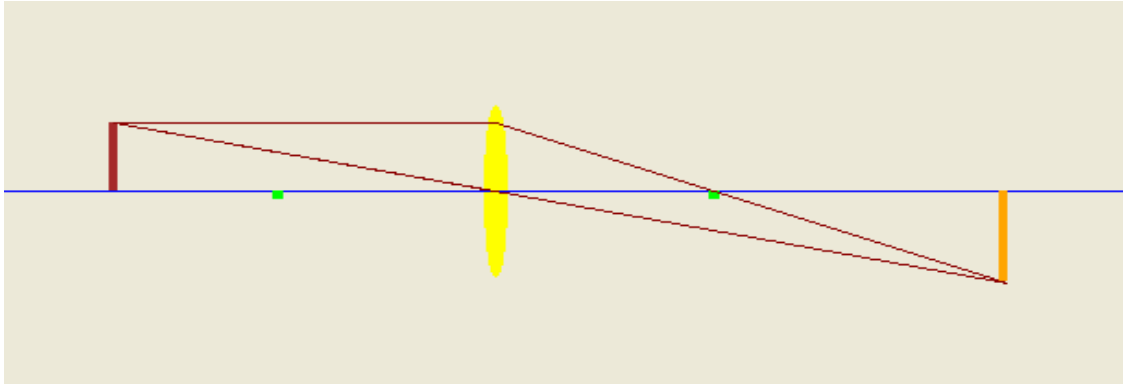

Figure 1. Ray diagram of a convex lens

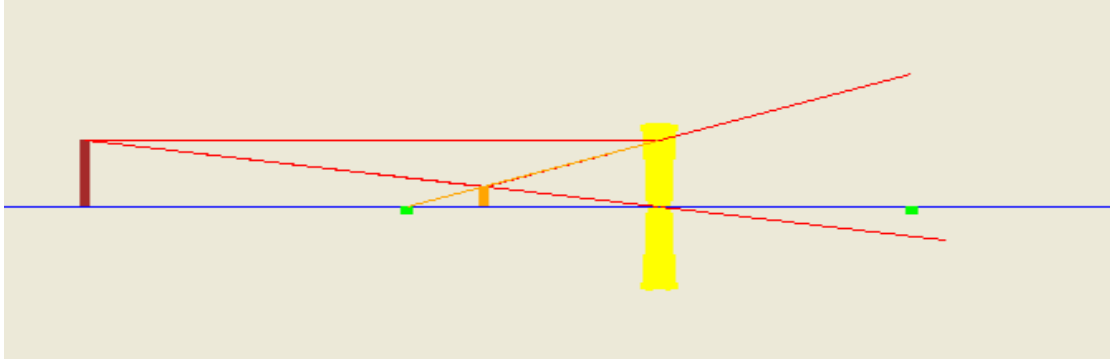

Figure 2. Ray diagram of a concave lens

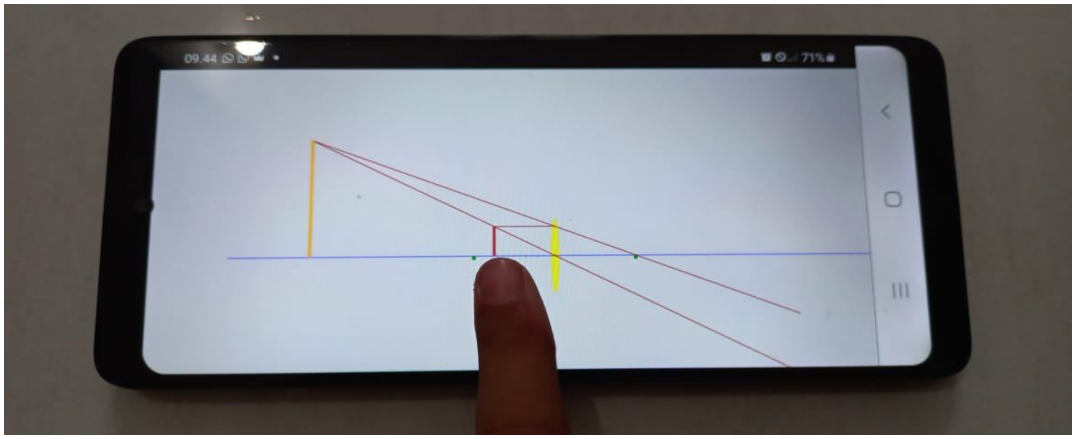

Figure 3. Display on an Android mobile phone

From the simulation results of the ray diagram simulation for the lens as shown in Figure 1. (convex lens) and Figure 2. (concave lens) it can be seen that the display of the program results consists of a lens, object and image as well as ray diagrams to obtain images of objects. By using the mouse, the cursor can be directed to move objects left or right. The shadow distance and height will automatically change according to the given formula. The display using a cellular phone is shown in Figure 3. To move an object, you do not need to use the mouse, but simply by touching and sliding with your fingers.

V. CONCLUSION

From the simulation results of ray diagrams on thin convex or concave lenses, it can be concluded that the ray diagram simulation program for lenses can be carried out well and can support the visual laboratory. To create visual simulations using the Python programming language, the canvas, create line, button, and motion functions can be used. The simulation can run correctly according to theory, and can be done easily using a cell phone. For further suggestions, we can perform simulations using more than one lens.

REFERENCES

1. White Jr, K.P. & Ingalls, R.G, 2009, Introduction to Simulation. Conference Paper in Proceedings Winter Simulation Conference. DOI: 10.1109/WSC.2009.5429315. https://doi.org/10.1109/wsc.2009.5429315
2. Gould, H. & Tobochnik, J., 1998, "Computer Simulation Method", Addison Wesley, New York.
3. Maria, A., 1997, Introduction to modeling and simulation, Proceedings of the 29th conference on Winter Simulation, p. 7–13. https://doi.org/10.1145/268437.268440
4. Hatherly, P.A., 2009, Interactive Screen Experiments - Innovative Virtual Laboratories for Distance Learners, European Journal of Physics 30(4):751. https://doi.org/10.1088/0143-0807/30/4/008
5. Hamed, G. & Aljanazrah, A., 2020, The Effectiveness of Using Virtual Experiments on Students' Learn-Ing in The General Physics Lab. Journal of Information Technology Education: Research, 19, 976-995. https://doi.org/10.28945/4668
6. Gonzalez, J.D., Escobar, J.H., Sanchez, H., De la Hoz1, J. & Beltran, J.R., 2017, 2D and 3D Virtual Interactive Laboratories of Physics on Unity Platform, Journal of Physics: Conf. Series 935 (2017) 012069. https://doi.org/10.1088/1742-6596/935/1/012069
7. Sand, D., 2014, Lenses,Technical Report, Birmingham City University. DOI: 10.13140/RG.2.1.4136.0247. https://www.researchgate.net/publication/302590143
8. Shaw, Z.A., 2014,Learn Python the Hard Way, Third Edition., Addison-Wesley, Upper Saddle River, NJ.
9. Beazley, D. & Jones, B., 2013, Python Cookbook, Third Edition, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
10. Jan Erik Solem, 2012, Programming Computer Vision with Python, Creative Commons Attribution-Noncommercial, http://creativecommons.org/licenses/by-nc-nd/3.0/us/