Journal of the Arkansas Academy of Science

Volume 48 Article 24

1994

Visualizing Electrostatic Phenomena Using Mathematica

Eric Mayes
Arkansas State University

Follow this and additional works at: https://scholarworks.uark.edu/jaas

 Part of the Numerical Analysis and Scientific Computing Commons, and the Programming Languages and Compilers Commons

Recommended Citation

Mayes, Eric (1994) "Visualizing Electrostatic Phenomena Using Mathematica," *Journal of the Arkansas Academy of Science*: Vol. 48 , Article 24.
Available at: https://scholarworks.uark.edu/jaas/vol48/iss1/24

This article is available for use under the Creative Commons license: Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0). Users are able to read, download, copy, print, distribute, search, link to the full texts of these articles, or use them for any other lawful purpose, without asking prior permission from the publisher or the author.
This Article is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Journal of the Arkansas Academy of Science by an authorized editor of ScholarWorks@UARK. For more information, please contact scholar@uark.edu.

Visualizing Electrostatic Phenomena Using Mathematica

Eric Mayes
Department of Computer Science, Mathematics and Physics
Arkansas State University
State University, AR 72467

Abstract

A set of packages for visualizing electrostatic phenomena was developed using *Mathematica* as a programming language. These packages allow users to plot potential fields, equipotential lines, 2-D and 3-D vector fields in order to gain a visual understanding of electrostatic charges. They would be useful in accompanying undergraduate physics labs pertaining to electrostatics, as they would enable students to connect experiment with mathematics through open-ended visual exploration.

Introduction

Mathematica (Wolfram, 1991) is a symbolic mathematics computer program that was first released in 1988. It can perform symbolic calculations, produce beautiful graphics, and give very accurate results limited only by the system that it is run on. Its most powerful feature, though, is that it is also a programming language. The purpose of this project was to visualize electrostatic phenomena and *Mathematica's* easily utilized, yet powerful, graphics abilities made it a perfect choice. The project is open-ended, in that others can easily create new change distributions and view them.

Theory

To produce the images of the electrostatic potentials and the electric fields, explicitly forms of the electric field and potential were used (Wangsness, 1986). The electric field is give by:

$$\mathbf{E}(\mathbf{r}) = \sum_{i=1}^{N} \frac{q_i \, \hat{\mathbf{R}}_i}{4\pi\epsilon_0 \, \mathbf{R}_i^2}$$

or more explicitly by:

$$\mathbf{E}(\mathbf{r}) = \sum_{i=1}^{N} \frac{q_i \left[(x-x_i)\hat{\mathbf{x}} + (y-y_i)\hat{\mathbf{y}} + (z-z_i)\hat{\mathbf{z}}\right]}{4\pi\epsilon_0 \left[(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2\right]^{\frac{3}{2}}}$$

Similarly the scalar potential is given explicitly by:

$$\phi(\mathbf{r}) = \sum_{i=1}^{N} \frac{q_i}{4\pi\epsilon_0 \left[(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2\right]^{\frac{1}{2}}}$$

These forms of the equations are easily manipulated using *Mathematica's* symbolic processing. A "Do Loop" adds up the contributions from each source point. For the electric field, a vector field is created; and for the electrostatic potential, a scalar field is created. These fields, through further manipulation, can be displayed on the screen or output in several forms, including printing to Post Script based laser printers.

Electrostatics Package

The first package (Listing 1), named "ElectroStatics," is loaded into *Mathematica Version 2.0* as:

```
In [1]:=<<ElectroStatics
```

There are four basic commands in this package. Two prompt the user for the charge distribution, and the other two accept list data types. The first command is "MakePField." This command is executed at the prompt in the form:

```
In[2]:=MakePField[N, {x, xmin, xmax}, {y, ymin, ymax},
(other opts)]
```

where "N" is the number of charger's in the distribution, "xmin, xmax, ymin, and ymax" are the bounds on the graph to be created and "other opts" specify other standard *Mathematica* graphics options. This command will return a 3-D surface. The Z-axis of the surface represents the magnitude of the electrostatic potential. Positive Z-axis values represent positive potentials, as negative values represent negative potentials. Note that *Mathematica* will chop off the tops and bottoms of charge's potentials as they diverge at the charge's locations. As an example:

```
In[2]:=MakePField[2, {x, -1, 2}, {y, -1, 2}, PlotPoints -> 40]
```

will prompt the user with:

```
Charge #1
-----------------
Charge: ?
```

The user can then enter the magnitude. The magnitude is entered in electrostatic units instead of Coulombs to keep the calculations simpler and faster. The user also enters the position of each charge up to "N" charges.

The second command is "MakeEField." This command takes the same form as the first, only it returns a plot of the electric field instead of the potential.

The third and fourth commands perform the same tasks as the first two, only they do not prompt the user for the magnitude and position. They expect to receive data of this form:

```
{charge, x coord, y coord, ...chargeN, xN coord, yN coord, N}
```

As an example:

```
In[3]:= PlotPField[{-1,1,1,1,-1,1,2}, {x,-2,2}, {y,-2,2}, PlotPoints -> 40]
```

These two commands were made so that the user could graph a function that would specify a specific charge distribution.

Figures 1a and 1b were created with the MakePField command, with N = 2. Figures 2a and 2b were created by displaying the result of 1a and 1b as contour plots using the standard *Mathematica* command:

```
Show [ContourGraphics["1a or 1b"], ContourShading -> False]
```

These plots are the equipotential lines for the charge distributions. Figures 3a and 3b were created for the same charge distribution with the MakeEField command. Figures 4a and 4b are combinations of 2a, 2b and 3a, 3b showing the electric field lines perpendicular to the equipotential lines. The third and fourth list input commands allow the user to display charge configurations generated by a function, rather then entering each charge individually.

As one way to utilize these commands, we can introduce a function like this:

```
RingOCharge[{x0_, y0_}, radius_, points_] :=
  Block[{i, theta, charges = {}},
    Do[AppendTo[charges,1];
    AppendTo[charges, (radius*Cos[theta] + x0)];
    AppendTo[charges, (radius*Sin[theta] +y0)];
    {theta, 0, 2*Pi, (2*Pi/points)}];
    AppendTo[charges, points]; charges]
```

It should be noted that the RingOCharge function sets the magnitude of each charge to 1. This function, when called with x0 and y0 being the center, will create a *Mathematica* list representing a ring of radius "radius" and approximated by "points" number of charges.

Figure 5 uses the third function, PlotPField with a list representing a ring of radius 3 centered at {0,0} approximated by 20 points. Figure 6 uses the fourth function, PlotEField with the same list.

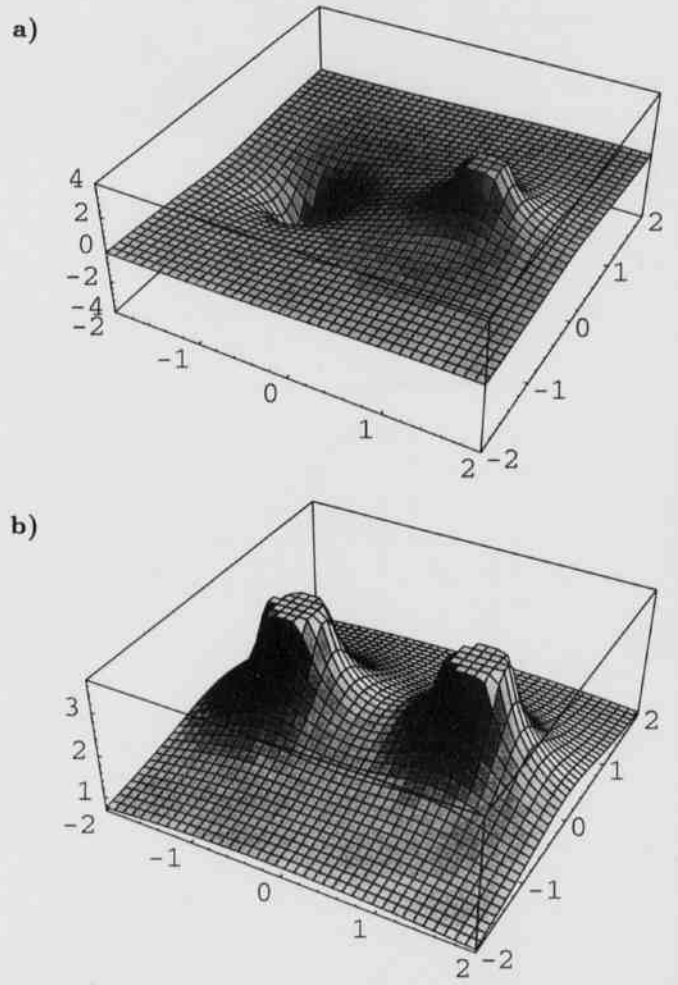

Fig. 1. a) Potential Energy Field for a Dipole [Inputs -1 at {-1,0} and 1 at {1,0}], b) Potential Energy Field for a Pair of Like Charges.

Electrostatics3D Package

The second package, "ElectroStatics3D," is loaded in the same manner:

```
In[1]:=<<ElectroStatics3D
```

There are only two commands in this package, one for

https://scholarworks.uark.edu/jaas/vol48/iss1/24

user input and the other for list input. Both commands create a 3-D plot of the electric field surrounding a set of charges. The first command, MakeEField3D, is called in almost the same manner as the MakeEField command, with the addition of a Z-coordinate:

In[2]:=MakeEFields3D[N, {x, xmin, xmax}, {y, ymin, ymax}, {z, zmin, zmax}, (other opts)]

Figure 7 is an example of this command with four co-planar charges.

Figure 8 demonstrates the second command in this package along with a slightly modified RingOCharge function in which the Z-coordinate was kept constant. This graph was produced from a list and the PlotEField3D command.

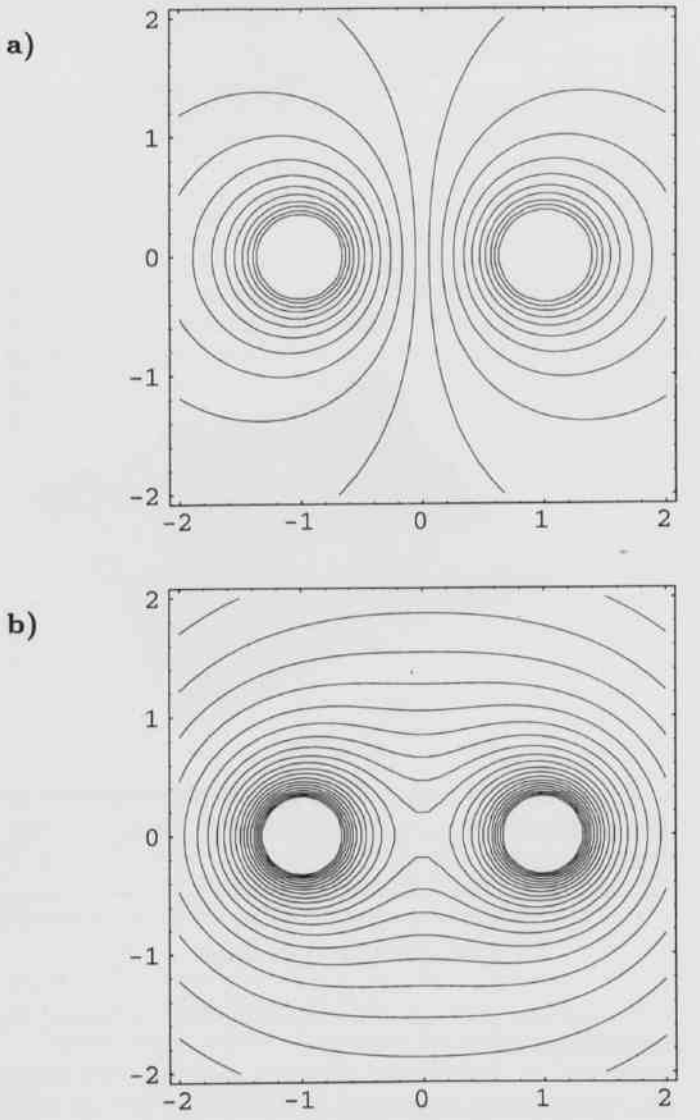

Fig. 2. a) Equipotential lines for a dipole, b) Equipotential lines for a pair of like charges.

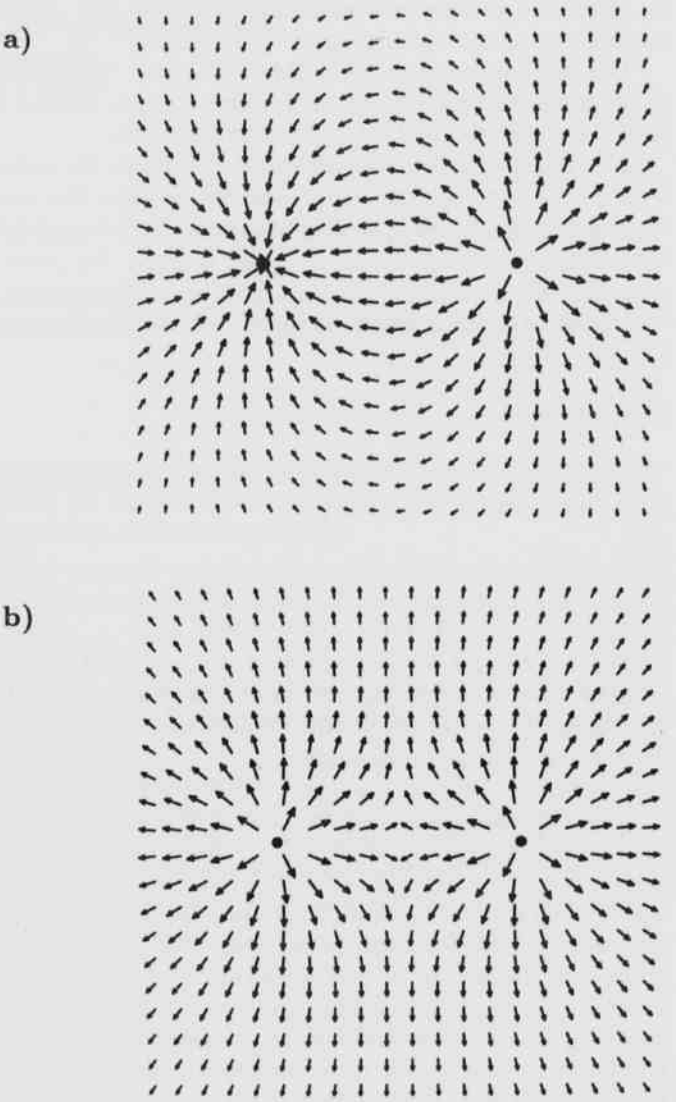

Fig. 3. a) Electric field vectors for a dipole, b) Electric field vectors for a pair of like charges.

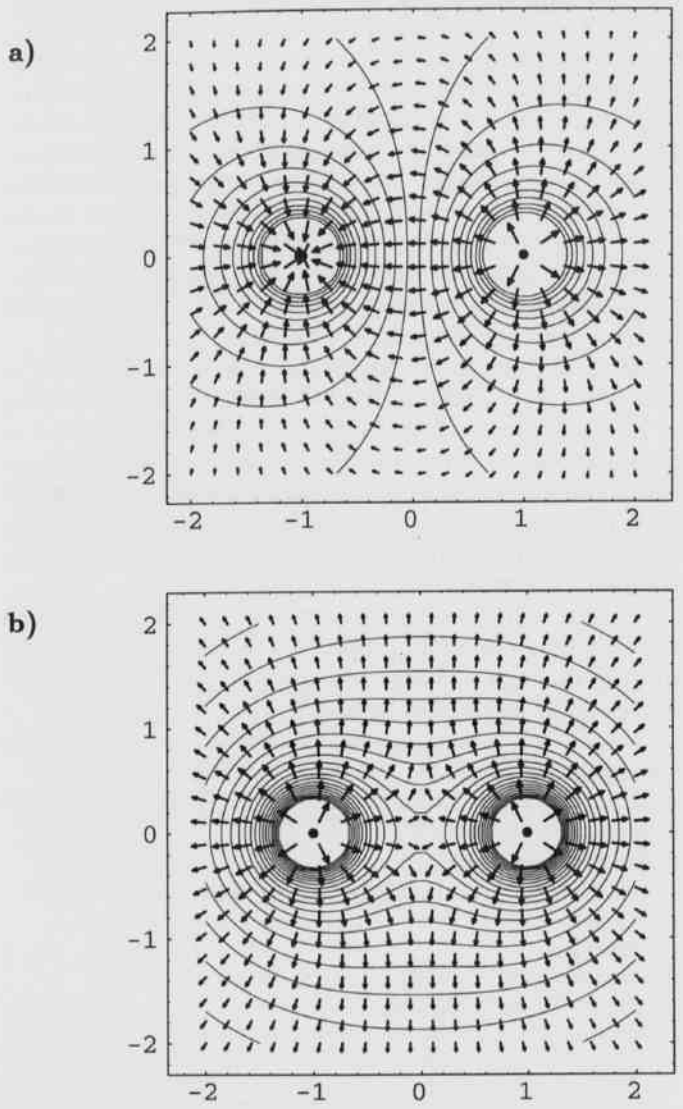

Fig. 4. Combination of equipotential lines and electric field vectors for two systems of charge. a) Dipole, b) Pair of like charges.

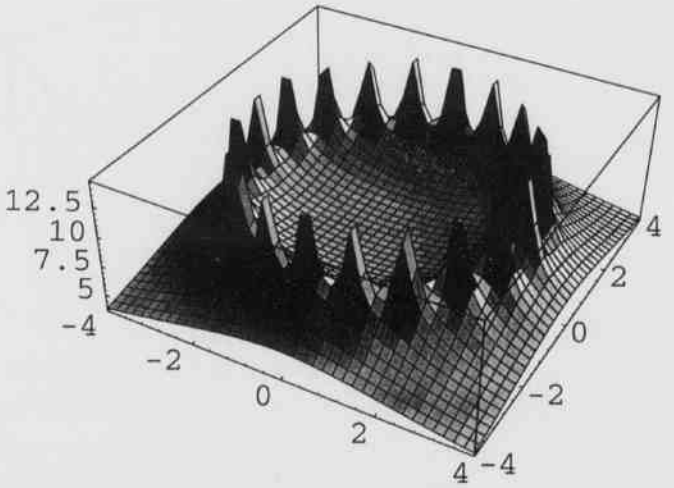

Fig. 5. Potential energy field for a ring of charge approximated by 20 point charges.

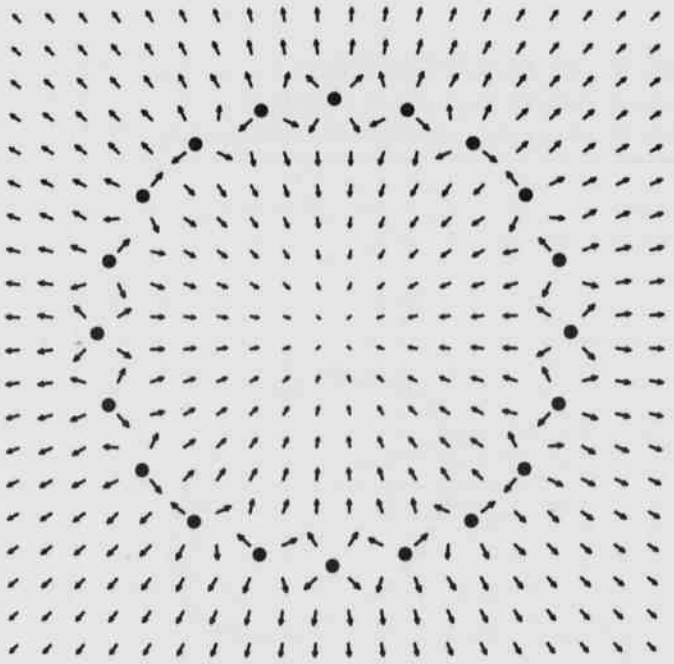

Fig. 6. Electric field vectors for a ring of charge approximated by 20 point charges.

https://scholarworks.uark.edu/jaas/vol48/iss1/24

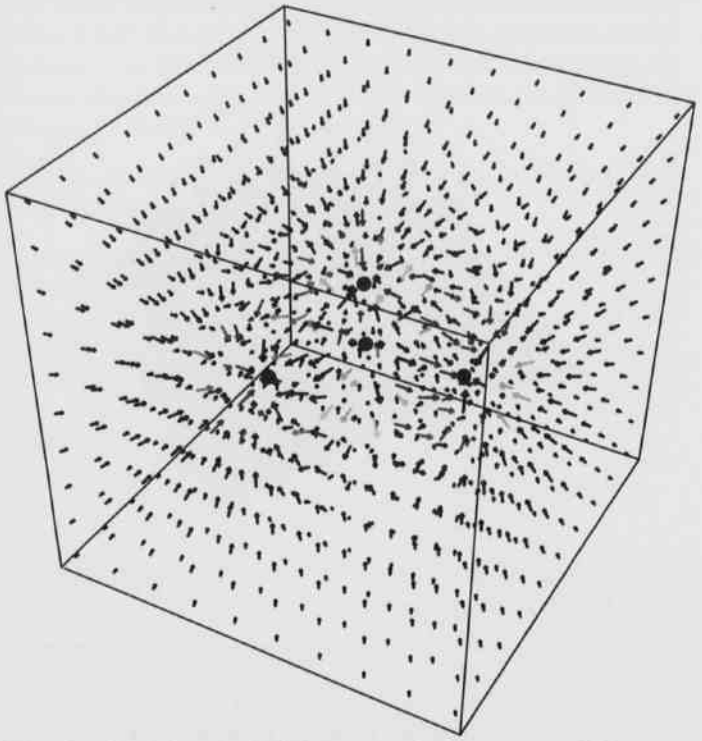

Fig. 7. 3-D Electric field vectors for a system of four coplanar charges [inputs 2 at {0,0,0}, -1 at {0,1,0}, {-0.866, -0.5, 0} where the -1 charges are separated by 120°].

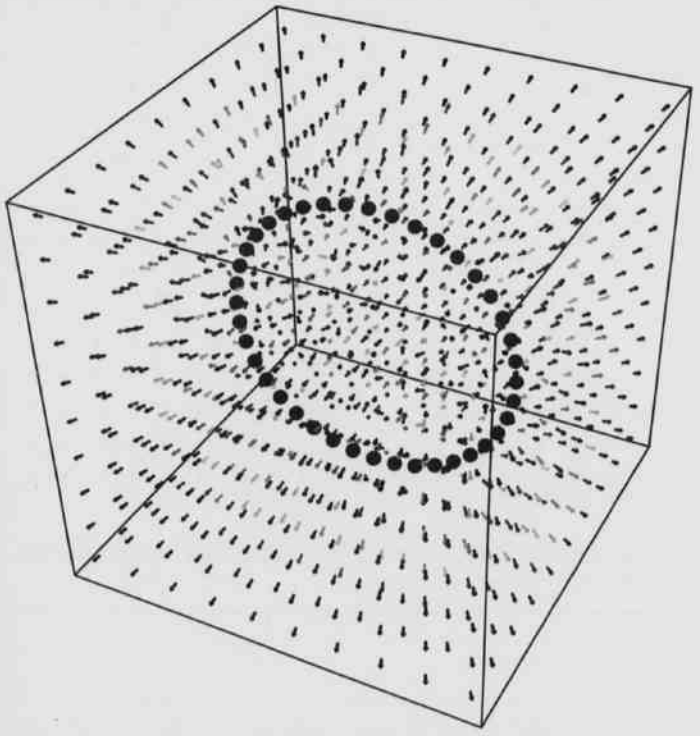

Fig. 8. 3-D electric field vectors for a ring of charge in the X-Z plane approximated by 40 point charges.

Discussion

These packages (See Listing 1 for ElectroStatics and Listing 2 for ElectroStatics3D) can be of educational value if combined with undergraduate physics labs pertaining to electrostatics. For labs that require the experimental mapping of equipotential lines, students could also model the lines on a computer using *Mathematica* and compare results. The packages may make learning electrostatics fun: making interesting charge distributions just to see what happens can be entertaining. *Programming in Mathematica* (Maeder, 1990) and *Mathematica, a System for Doing Mathematics by Computer* (Wolfram, 1991) are good references to help with such a lab.

Modeling complex distributions is time intensive. A Sun SPARC-IPC, with a SPARC RISC processor running at 15.8 MIPS and 1.7 MFLOPS, took about 35 minutes to complete Fig. 8. Unfortunately, *Mathematica* packages are not stand-alone. *Mathematica* has its own kernel which must be present in order to run the packages; so economic concerns could limit the packages' use, as *Mathematica* is rather expensive.

Future work on this project will utilize *Mathematica 2.0's* ability to animate graphics. Hopefully a test charge moving through a charge distribution can be modeled in a reasonable amount of time.

Listing 1. The Code for the "ElectroStatics" Package.

```
(*:Version: Mathematica 2.0*)
(*:Name: ElectroStatics' *)
(*:Title: Electric Potential and Field Plots for Static Charges *)
(*: Author:
    Eric Mayes
    Arkansas State University
    emayes@quapaw.astate.edu *)
(*:Summary: This package allows the user to input a set of static
    charges on the x-y plane. The results, depending on the selected
    function, will display either a graph of the magnitude of the electric
    potential over a region; or it will display a 2-D vector plot of the
    electric field. *)
BeginPackage["ElectroStatics'", "Graphics'PlotField'"]

MakePField::usage = "MakePField[N, {x,xmin,xmax}, {y,ymin,ymax},
    (other opts)] returns a 3-D graph of the potential field of a set of N
    charges specified by the user.";

MakeEField::usage = "MakeEField[N, {x,xmin,xmax}, {y,ymin,ymax},
    (other opts)] returns a graph of the electric field of a set of N
    charges specified by the user.";

PlotPField::usage = "PlotPField[list, {x,xmin,xmax}, {y,ymin,ymax},
    (other opts)] returns a 3-D graph of the potential field of a list of N
    charges that conform to the format {charge1,x1,y1...,chargeN,xN,yN
    ,N}.";

PlotEField::usage = "PlotEField[list, {x,xmin,xmax}, {y,ymin,ymax},
    (other opts)] returns a graph of the electric field of a list of N
    charges that conform to the format
```

```
    {charge1,x1,y1...,chargeN,xN,yN,N}.";

Begin["`Private`"]

MakePField[num_, {x0_, x1_, x2_}, {y0_, y1_, y2_}, opts___] :=
    Block[{i, chargelist = {}, potential = 0},
     Do[
        Print [" "];
        Print["Charge #",ToString[i]];
        Print["———"];
        answer = Input["Charge: "];
          AppendTo[chargelist,answer];
        answer = Input["X Coord: "];
          AppendTo[chargelist,answer];
        answer = Input["Y Coord: "];
          AppendTo[chargelist,answer],
    {i,num}];
    AppendTo[chargelist,num];
    Do[
        potential = potential + (Part[chargelist,i]/
        ((x0 - Part[chargelist, (i+1)])^2 +
          (y0 - Part[chargelist, (i+2)])^2)^.5),
    {i,1,(Last[chargelist]*3),3}];
    Plot3D[potential, {x0, x1, x2}, {y0, y1, y2}, opts]]

MakeEField[num_, {x0_, x1_, x2_}, {y0_, y1_, y2_}, opts___] :=
    Block[{i, vectorplot, dotsplot, chargelist = {}, dots = {}, field = 0},
     Do[
        Print[" "];
        Print["Charge #",ToString[i]];
        Print ["———"];
        answer = Input["Charge: "];
          AppendTo[chargelist, answer];
        answer = Input["X Coord: "];
          AppendTo[chargelist, answer];
        answer = Input["Y Coord: "];
          AppendTo[chargelist, answer],
    {i,num}];
    AppendTo[chargelist, num};
    Do[(field = field + ((Part[chargelist,i]*
        ({(x0 - Part[chargelist, (i+1)]),
          (y0 - Part[chargelist, (i+2)])}))/
        ((x0 - (Part[chargelist, (i+1)]+.0001))^2 +

    (* .0001 Added To Protect From Division By Zero *)

        (y0 - (Part[chargelist, (i+2)]+.0001))^2)^(3/2))),
    {i,1,(List[chargelist]*3),3}];
    vectorplot = PlotVectorField[field, {x0, x1, x2}, {y0, y1, y2}, opts,
    ScaleFunction->(Log[Log[Log[Log[Log[Log[#+1]+1]+1] +1]+1]+1]&),
    ScaleFactor -> .2, DisplayFunction -> Identity];

    Do[AppendTo[dots,Point[{Part[chargelist, (i+1)], Part [chargelist,
        (i+2)]}]], {i,1,(Last[chargelist]*3),3}];
    dotsplot = Graphics[{PointSize[0.02], dots}, Display Function->
        Identity];
    Show[vectorplot,dotsplot,DisplayFunction -> $Display Function]]

PlotPField[chargelist_, {x0_, x1_, x2_}, {y0_, y1_, y2_}, opts___] :=
    Block[{i, potential =0},
     Do[
        potential = potential + (Part[chargelist,i]/
        ((x0 - Part[chargelist,(i+1)]) ^2 +
         (y0 - Part[chargelist,(i+2)])^2)^.5),
    {i,1,(Last[chargelist]*3),3}];
    Plot3D[potential, {x0, x1, x2}, {y0, y1, y2}, opts]]

PlotEField[chargelist_, {x0_, x1_, x2_}, {y0_, y1_, y2_}, opts___] :=
    Block[{i, vectorplit, dotsplot, dots = {}, field = 0},
     Do[
        field = field + ((Part[chargelist,i]*
        ({(x0 - Part[chargelist, (i+1)]),
          (y0 - Part[chargelist, (i+2)])}))/
        ((x0 - (Part[chargelist,(i+1)]+.0001))^2 +
          (y0 - (Part[chargelist,(i+2)]+.0001))^2)^(3/2)),
    {i,1,(Last[chargelist]*3),3}];
    vectorplot = PlotVectorField[field, {x0, x1, x2}, {y0, y1, y2}, opts,
    ScaleFunction->(Log[Log[Log[Log[Log[Log[#+ 1]+1]+1]+1]+1]+1]&),
    ScaleFactor -> .2, DisplayFunction -> Identity];

    Do[AppendTo[dots,Point[{Part[chargelist, (i+1)], Part [chargelist,
        (i+2)]}]], {i,1,(Last[chargelist]*3),3}];
    dotsplot =Graphics[{PointSize[0.02], dots},DisplayFunction ->
        Identity];
    Show[vectorplot,dotsplot,DisplayFunction ->  $DisplayFunction]]

End [] (* ElectroStatics`Private` *)

Protect[MakePField, MakeEField, PlotPField, PlotEField]

EndPackage[] (* ElectroStatics` *)
```

Listing 2. The Code for the "ElectroStatics3D" Package.

```
(*:Version: Mathematica 2.0 *)

(*:Name: ElectroStatics3D` *)

(*:Title: 3-D Electric Field Plots for Static Charges *)

(*:Author:
    Eric Mayes
    Arkansas State University
    emayes@quapaw.astate.edu *)

(*:Summary: This packages allows the user to input a set of static
charges in 3-space. The result will be displayed in a 3-D vector field plot.
*)

BeginPackage["ElectroStatics3D`", "Graphics`PlotField3D`"]

MakeEField3D::usage="MakeEField3D[N, {x,xmin,xmax},{y,ymin,ymax},
    {z,zmin,zmax}, (other opts)] returns a 3-D graph of the electric
    field of a set of N charges specified by the user."

PlotEField3D::usage ="PlotEField3D[list, {x,xmin,xmax},{y,ymin,ymax},
    {z,zmin,zmax}, (other opts)] returns a 3-D graph of the electric
    field of a list of N charges that conform to the format
    {charge1,x1,y1,z1...,chargeN,xN,yN,zN,N}."

Begin["`Private`"]

MakeEField3D[num_, {x0_, x1_, x2_}, {y0_, y1_, y2_}, {z0_,  z1_, z2_},
        opts___] :=
    Block[{i, vectorplot, dotsplot, dots = {}, chargelist = {}, field = 0},
    Do[
        Print[" "];
        Print["Charge #",ToStrin[i]];
        Print["———"];
        answer = Input["Charge: "];
          AppendTo[chargelist,answer];
        answer = Input["X Coord: "];
          AppendTo[chargelist,answer];
        answer = Input["Y Coord: "];
          AppendTo[chargelist,answer];
        answer = Input["Z Coord: "];
          AppendTo[chargelist,answer],
```

```
    {i,num}];
    AppendTo[chargelist,num];
    Do[
        field = field + ((Part[chargelist,i]*
        ({(x0 - Part[chargelist, (i+1)]),
          (y0 - Part[chargelist, (i+2)]),
          (z0 - Part[chargelist, (i+3)])})/
        ((x0 - (Part[chargelist,(i+1)]+.0001))^2 +

        (*.0001 Added To Protect From Division By Zero *)

        (y0 - (Part[chargelist,(i+2)]+.0001))^2 +
        (z0 - (Part[chargelist,(i+3)]+.0001))^2)^(3/2)),
    {i,1,(Last[chargelist]*4),4}];
    vectorplot = PlotVectorField3D[field, {x0, x1, x2}, {y0, y1, y2},
    {z0, z1, z2}, opts, ScaleFunction -> (Log[Log[Log[Log[Log[Log[# +
     1]+1]+1]+1]+1]+1]&), ScaleFactor -> .2, DisplayFunction -> Identity];

    Do[AppendTo[dots,Point[{Part[chargelist,(i+1)],Part[chargelist, (i+2)],
        Part[chargelist, (i+3)]}]], {i,1,(Last[chargelist]*4),4}];
    dotsplot = Graphics3D[{PointSize[0.02], dots},DisplayFunction ->
        Identity];
    Show[vectorplot,dotsplot,DisplayFunction -> $DisplayFunction]]

PlotEField3D[chargelist_, {x0_, x1_, x2_}, {y0_, y1_, y2_}, {z0_, z1_, z2_},
        opts___] :=
    Block[{i, vectorplot, dotsplot, dots = {}, field = 0},
      Do[
        field = field + ((Part[chargelist,i]*
        ({(x0 - Part[chargelist, (i+1)]),
          (y0 - Part[chargelist, (i+2)]),
          (z0 - Part[chargelist, (i+3)])})/
        ((x0 - (Part[chargelist, (i+1)]+.0001))^2 +
         (y0 - (Part[chargelist,(i+2)]+.0001))^2 +
         (z0 - (Part[chargelist,(i+3)]+.0001))^2)^(3.2)),
    {i,1,(Last[chargelist]*4),4}];
    vectorplot = PlotVectorField3D[field, {x0, x1, x2}, {y0, y1, y2},
    {z0, z1, z2}, opts, ScaleFunction -> (Log[Log[Log[Log[Log[Log[# +
    1]+1]+1]+1]+1]+1]&), ScaleFactor -> .2, DisplayFunction -> Identify];

    Do[AppendTo[dots,Point[{Part[chargelist, (i+1)], Part[chargelist,
      (i+2)], Part[chargelist, (i+3)]}]], {i,1,(Last[chargelist]*4),4}];
    dotsplot = Graphics3D[{PointSize[0.02], dots},DisplayFunction ->
        Identity];
    Show[vectorplot,dotsplot,DisplayFunction -> $DisplayFunction]]

End [] (* ElectroStatics3D`Private` *)

Protect[MakeEField3D, PlotEField3D]

EndPackage[] (* ElectroStatics3D` *)
```

Acknowledgements

This work acknowledges Charles A. Hughes for his assistance and the Department of Computer Science, Mathematics, and Physics at Arkansas State University for the use of their facilities.

Literature Cited

Wangsness, R.K. 1986, Electromagnetic Fields, 2nd ed. John Wiley & Sons, Inc., New York, Pp. 51-82.

Maeder, R. 1990. Programming in Mathematica. Addison-Wesley, Redwood City, California.

Wolfram, S. 1991. Mathematica, a System for Doing Mathematics by Computer. Addison-Wesley, Redwood City, California.