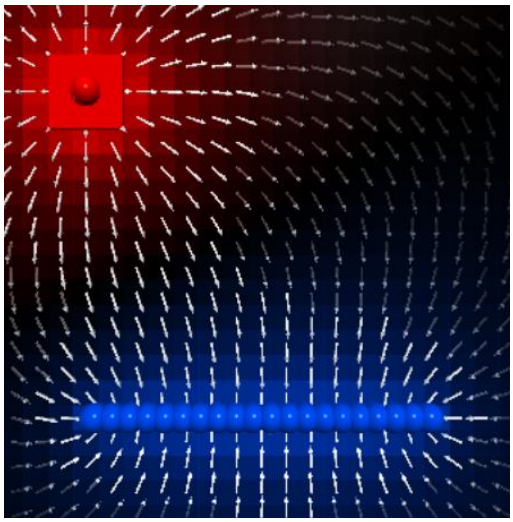
Potential & Electric Field Visualization

Your ultimate goal is to produce a code which correctly uses physics to simultaneously visualize electric field and electric potential due to an assembly of charges similar to the one shown below.

Each student is expected to submit original work. That said, it is often nice to work closely with a single partner. If you worked with a partner, you are expected to include your partner's name on your final submission (clearly indicating who is author and who is partner).

Your final code must include the following:

- A positive charge of $+1.00$ nC
- A negative charge of -1.00 nC
- One of the charges must be a distributed charge comprised of at least 20 pieces.
- Positive charges are colored red
- Negative charges are colored blue (with a little bit of green)
- Intensity of color indicates strength of electric potential
- Arrow opacity indicates strength of electric field



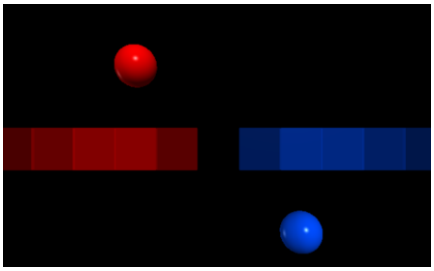
Upon completing the code, the last page of this document tells you what I expect you to submit. Instructions begin on the next page.

ATTENTION: in your write-up, you are required to submit pseudo code.
Why not take a few minutes to write out a pseudo code right now?

Don't try to be perfect.
Think through the major issues you might expect in trying to create the image above.
Imagine what input parameters you might need, what loops you might need, what functions you would nice to have.
Spend 5minutes cranking out your thoughts as quickly as possible.
If you've never written pseudo code, spend 5 minutes skimming these articles.

- First link about writing pseudocode.
- Another link about how to write pseudocode.

Consider the code shown on the right side of this page.
This code was used to generate the visualization of the electric potential shown below.



This first block of code shows the main constants and draws two point charges with opposite signs.

```
1  Web VPython 3.2
2
3  k = 8.99e9
4  Q = 1e-9
5  x_min = -5
6  x_max = 5
7  dx = 1
8  positive_color = vec(1, 0,0)
9  negative_color = vec(0, 0.3, 1)
10 V_max = 5   #saturation level for voltages
11
12 q_1 = sphere( pos = vec(-2,2,0),
13               radius = 0.5,
14               q = Q,
15               color = positive_color )
16
17 q_2 = sphere( pos = vec(2,-2,0),
18               radius = 0.5,
19               q = -1*Q,
20               color = negative_color )
```

Here is a function I defined. Unlike constants, it is ok to define functions at the *bottom* of the code even though you intend to use them in lines of code above them. I rather liked having constants at the top, functions at the bottom, and actual guts of the code in the middle.

I drew orange rectangles to obscure parts of the code so you would have to figure some stuff out yourself. It may help to read line 28 (see lowest figure at right) to see how I used the function.

```
44 #define functions
45 def get_V_contribution(charge, current_POI):
46     r = [redacted]
47     if r == 0:
48         V_contribution = 0
49     else:
50         V_contribution = [redacted]
51     return V_contribution
```

The guts of the code come in this FOR loop.
At first it might seem like overkill to write it this way. However, as the lab continues, I believe you will appreciate how easy it is to generalize this code to handle all that we want to do. Again, I drew some orange rectangles to obscure parts of the code so you would have to come up with a few things by yourself.

Important to notice: in general it is bad form to scale the actual voltage computed by the computer. We do, however, need to scale the output visualization of the voltage (in this case the plate opacity). Notice in lines 36 & 37 we affect the plate *opacity* if the total voltage is outside the saturation level limits but we never actually change the computed voltage.

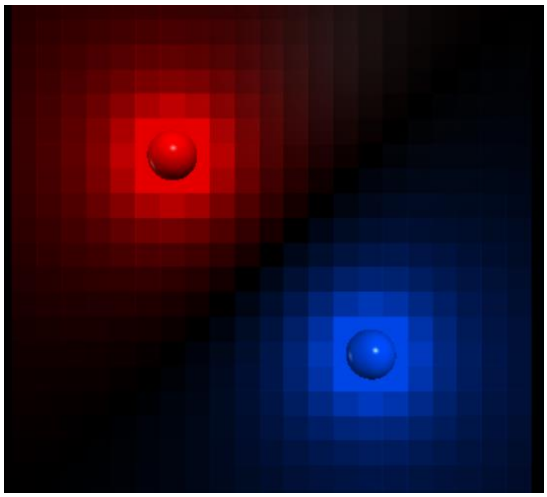
```
22 for x in arange(x_min, x_max + 0.5*dx, dx):
23     rate(4)
24
25     V_total = 0
26     POI = vec(x, 0, 0)
27
28     V_total += get_V_contribution(q_1, POI)
29     V_total += [redacted]
30
31     plate = box( pos = POI, size = vec(dx, dx, 0.1*dx) )
32     if V_total >= 0:
33         plate.color = [redacted]
34     else:
35         plate.color = [redacted]
36     if -1*V_max < V_total < V_max:
37         plate.opacity = abs(V_total)/V_max
38
```

This is an important distinction as later we may want to visualize the field (which requires a saturation level) while simultaneously using the underlying values to compute animations (which requires *no* saturation level).

Think: why are lines 47 & 48 necessary for this code?
Hint: What happens if you change the charge positions to have $y = 0$?

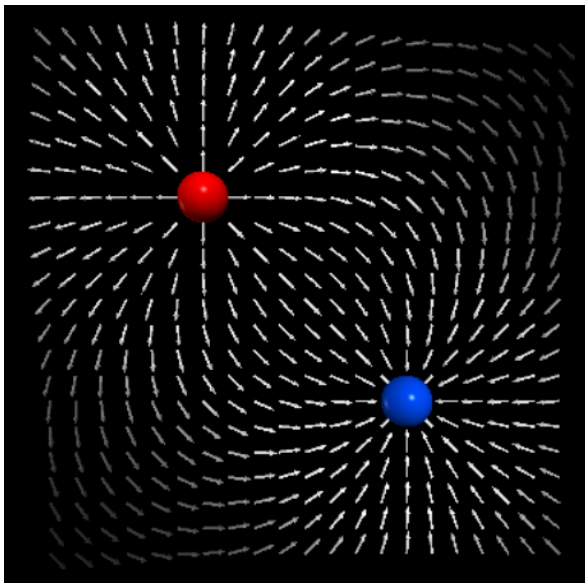
Next I generalized the 1D potential code to 2D.
Make a copy of your previous code before starting this one!
Think about someone reading your code (me) as you generalize it. It may seem like a waste of time to define $y_{min}, y_{max}, \& \, dy$ but it makes the code more intuitive for a reader.

The screenshot used $V_{max} = 10$ V and $dx = 0.5$ m.

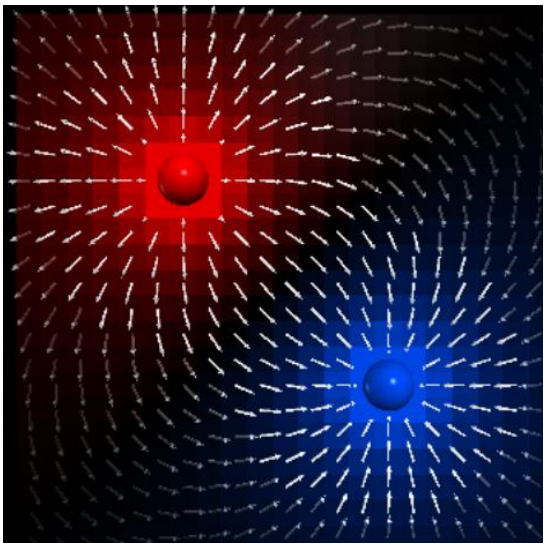


Next I modified the potential code to draw electric field arrows instead of plates representing electric potential.
Make a copy of your previous code before starting this one.
Challenges I encountered:

1. The screenshot used $E_{max} = 1$ V/m and $dx = 0.5$ m.
2. I had to create a different function for computing electric field contributions.
3. I changed my saturation level variable name to `E_max`. Remember: we eventually plan to put the codes together so you'll need clear & unique variable names.
4. The electric field computed at each POI needs to be a *vector* (to draw various directions for all the arrows). The saturation level is a *scalar*.
5. Instead of scaling the arrow *size* with the saturation level, I instead scaled arrow *opacity*. Remember: scale the *visualization* of the field, not the actual field.

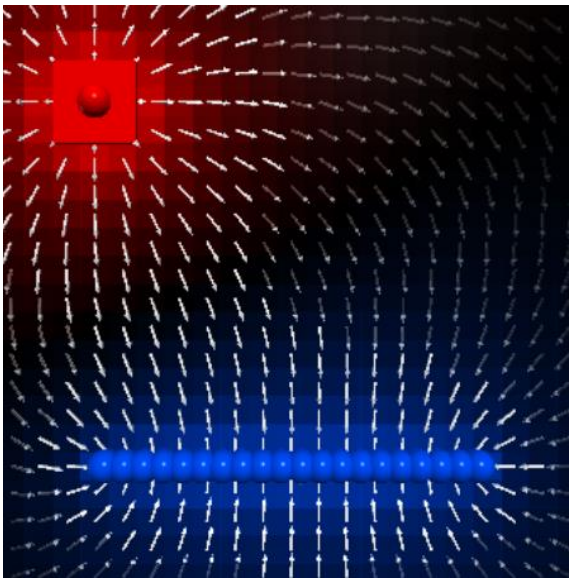


Next I put the two 2D codes together to get following screenshot.
Make a copy of your previous code before starting this one.
Here I used $V_{max} = 10\ \text{V},\ E_{max} = 1\ \text{V/m}, \& \ dx = 0.5\ \text{m}$.



Finally, we want to generalize the code to handle continuous distributions of charge.
Make a copy of your previous code before starting this one.

- Using $V_{max} = 10\ \text{V},\ E_{max} = 1\ \text{V/m}, \& \ dx = 0.5\ \text{m}$.
- Change radii of point charges to $0.3\ \text{m}$.
- Location of charge 1 is now $< -3.5, 3.5, 0 >\ \text{m}$. Still using $q_1 = +1\ \text{nC}$.
- Rod carries $-1\ \text{nC}$ distributed uniformly.
    - I created a new set of parameters such as `rod_x_min`, `rod_dx`, `rod_q`, etc. This reduced ambiguity when I had to add a new for loop to draw the rod.
    - I remembered to offset the charges in the rod using our old trick: `x_min+0.5*dx`
    - I used 20 total charges to build the rod, but I did all of my code testing using $N = 5$.
    - When testing I also set q_1 to zero to make it easier to verify I was correctly generating the field for charges in the rod.
    - In the for loop used to build the rod, I remembered to append each charge into a list. This dramatically simplified computing contributions to both $\vec{E}_{total}$ & V_{total}.



How will you be graded?

1. Each student is expected to submit a single pdf file. Include a lab title, your name (indicated as author), partners names, lab time should appear centered on of the first page. Making a title for real scientific work is an important skill but not something I want to talk about for this report…keep it simple.
2. A pseudo code outline of what your final code ideally should look like.
    - First link about writing pseudocode.
    - Another link about how to write pseudocode.
    - Consider dumping your final code into AI to have it produce the pseudo code for you, then read it and tweak it until it makes sense. Compare this to the pseudo code you wrote before starting…
3. Links to the all the codes you used. For each code, include a screen shot of the code output next to the link (like a thumbnail) to make it easier for me to check the work.
4. If you failed, explain which parts of the pseudocode were too difficult for you to complete. Mention this information near the links to your code in your final document.
5. Regardless of your ability to complete the project, include 2-3 paragraphs reflecting on your experiences writing this code. I've included some reflection questions below to help you get started on this reflection. You do not need to answer all of these questions, but hopefully they help you get some thoughts going so you can get two solid paragraphs (That said, you can take this in a different direction if that is where your reflections take you. Even though this seems cheesy, I've read that the reflection process is useful for locking in what you've learned. In that spirit, try to take it seriously.
    - How has programming in physics changed the way you program or view programming?
    - If you had more time, what are the next pieces of this particular code you would try to add?
    - Alternatively, what coding project might you try next in Python?
    - What do you notice you like/dislike about programming in Python?
    - Are there any obvious pros/cons to programming in Python versus other languages?
    - What are you going to put on your resume regarding your current level of Python fluency?
    - What are the next steps you need to take to increase your Python fluency? Be specific with your plan (i.e. include links to playlists). Who knows...you may actually do it!
    - Do you have any questions or thoughts of your own you'd like to share?