Technical Details for Applet

The technical details for the Equipotential Surfaces Applet provide an overview of the representation used in the applet and how the applet is controlled. This document also describes some of the more challenging algorithms implemented in the applet.

Background on Equipotential Surfaces

Equipotential Surfaces deal with charged particles in electric fields. Each particle in an electric field carries a positive or negative charge with it; these charges create a potential energy that is dispersed throughout the field. The amount of potential energy at a particular point in an electric field depends on the number of particles in a field, their charges, and the distance from the point to each particle. The potential energy (V) at a point is computed by the equation:

$$V = k\sum_{n} \frac{q_n}{r_n}$$

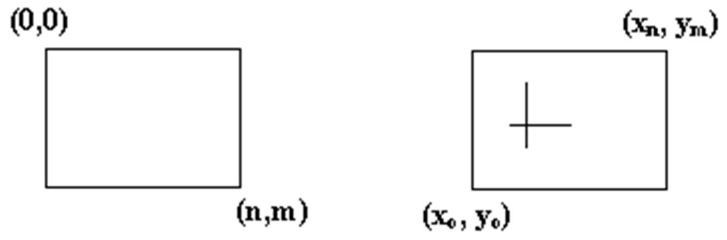
Here q_n is the charge of the nth particle and r_n is the radial distance of the point in question from the nth particle. The value of k is a constant used in computing the potential.

A locus of points all having the same potential (V) is called an equipotential surface. Calculating these by hand is a very tedious process, especially when the number of particles starts to grow. Through the use of this applet, one can quickly find equipotential surfaces in a field of a point charge.

Model Representation

The first object that needs to be represented is the electric field. This is done through the use of a canvas, or more specifically, the Electric Field object. The problematic item that needs to be addressed is the conversion of a computer canvas's coordinate system to an electric field's coordinate system. The canvas coordinate system (Figure 1) has a "upside-down" y-axis and ranges in integer coordinates from (0,0) at the top left of the canvas to (n, m) at the bottom right of the canvas, where n and m are integers. The points of the electric field (Figure 2) need to range from (x_o, y_o) at the bottom left-hand corner of the viewable field to (x_n, y_m) at the top right-hand corner of the viewable field, where x and y can be both positive or negative real numbers. This is done through the introduction of another object (WindowToCanvas) that handles the conversion of coordinates between the canvas system and the field system. The

WindowToCanvas object converts between the two coordinate systems by using an affine transform.



A Particle object represents the charged particles placed onto this field; it is used to keep track of the characteristics of a particular particle. For instance, it holds information about the particle's position on the field and the particle's charge. Since there can be any number (n) of particles on a given electric field, an NParticle object is introduced to keep track of the particles currently on the field.

The equipotential surface itself is represented by a Locus object; this object holds the points of the equipotential surface and the (equivalent) potential of those points. Similar to the particles, there can be up to *n* loci on an electric field. Thus, an NLocus object is used to maintain the loci on the electric field.

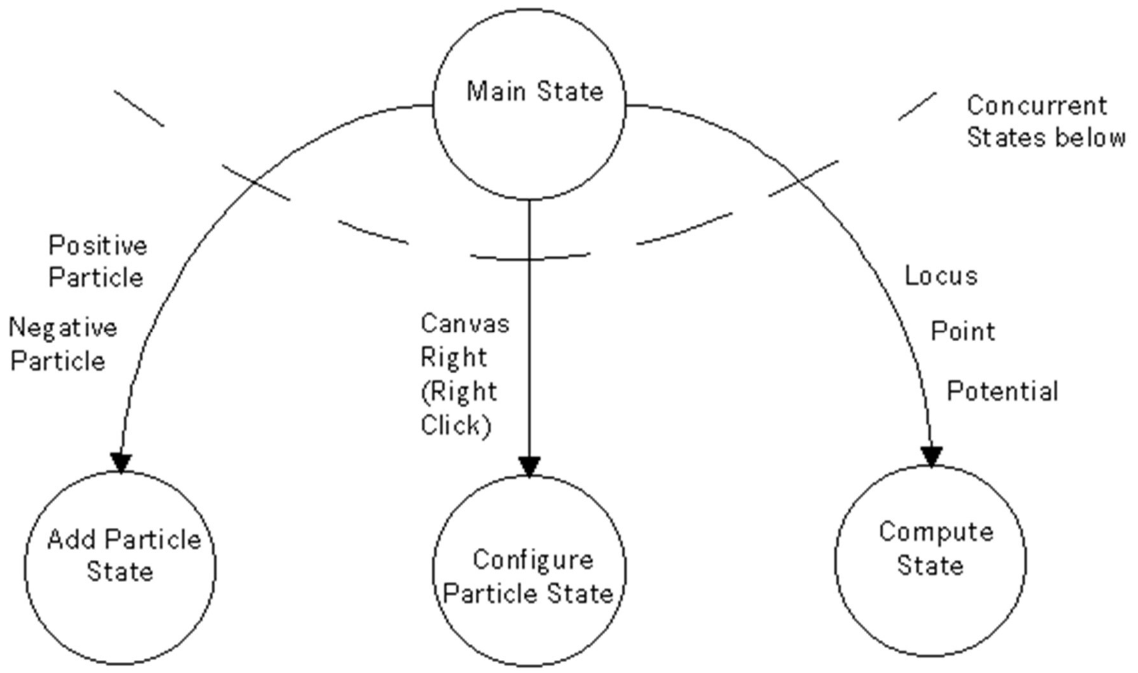
View Representation

The view for the model is the same canvas used to represent the electric field. It holds the NLocus and NParticle objects, handles putting a visual representation of these objects on the field whenever necessary, and keeps track of the size of the viewable area of the electric field. By use of the WindowToCanvas object, the viewable size of the field can be set to virtually any area. For example, the field's coordinate system can have a range from (-0.1, -0.1) to (1.2,1.12) or range from
(-1000, -1500) to (2000, 3000).

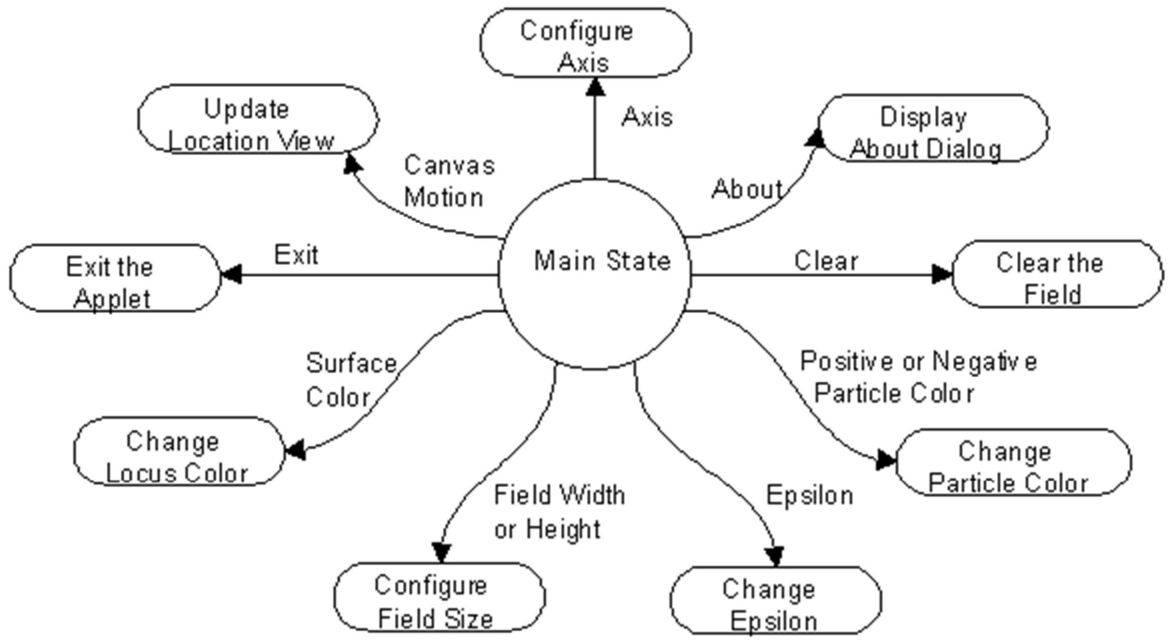
The canvas representing the electric field also has an x-y axis that is placed on it to give the user a reference point for the current field coordinate system.

Control Representation

The model and view are controlled by a state machine. The state machine listens for action events that occur on the user interface and responds appropriately to the action event. The applet opens up and stays in the Main State. Upon receiving notification that an action event has occurred, the Main State either handles the action event or moves into a sub-state that operates concurrently with the Main State. The state diagram below shows which action events cause the Main State to enter a concurrent sub-state. The name of the state is located inside the state symbol (circle), and the action event that moves from one concurrent state to another is written on the arrows between the states. There can only be one state operating concurrently with the Main State, so when a sub-state is entered, the existing concurrent state must be exited.

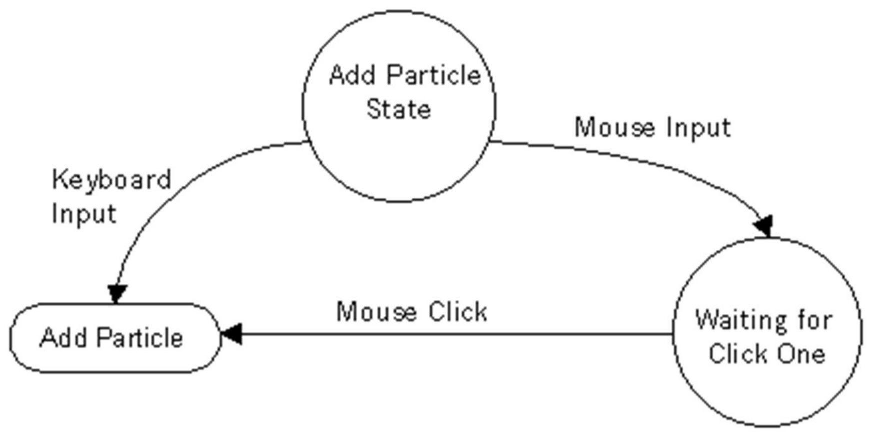


Now, previously mentioned is the fact that the Main State handles some action events. It mostly handles the events that do not deal directly with the model used to represent the equipotential surface or particle. The text on the arrow is the action event that causes a particular activity to take place. Inside the activity symbol (ellipses) is the name of the activity entered.

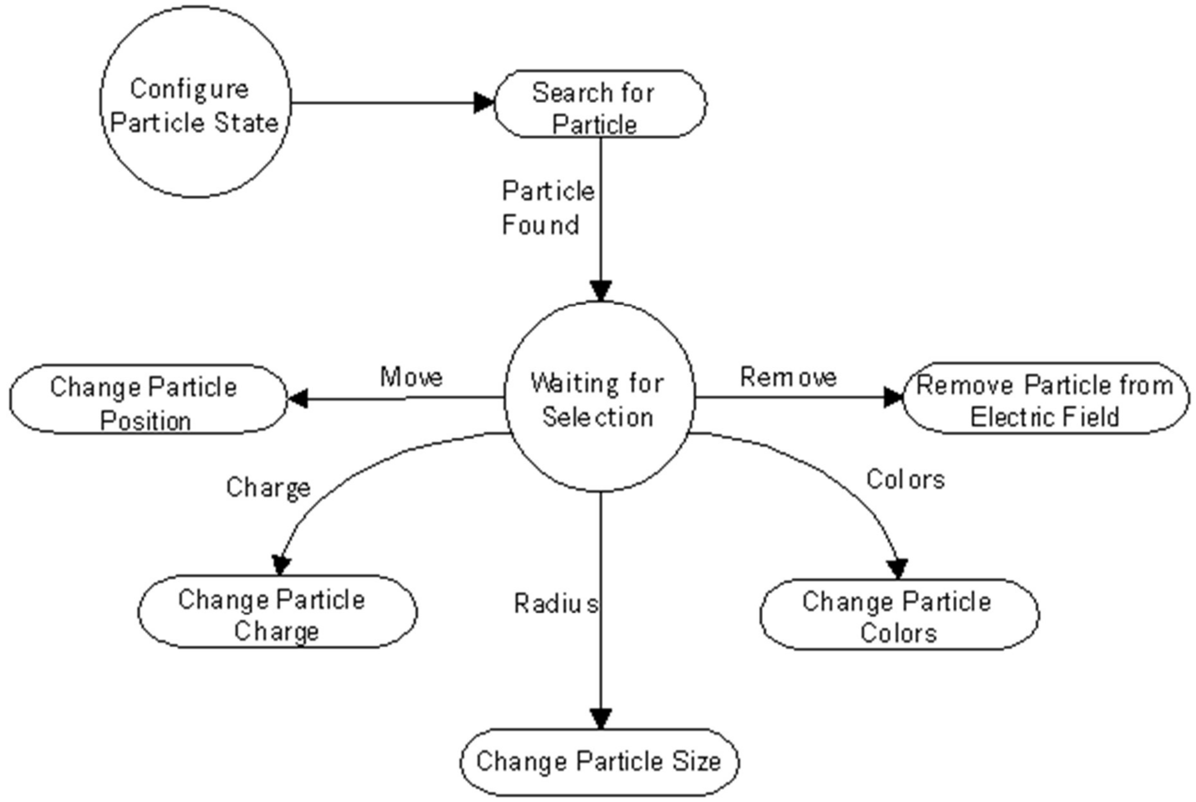


Each concurrent state also has certain activities that it handles. The Add Particle State adds a particle to the electric field. Since precision and accuracy are important in studying equipotential surfaces, the Add Particle State allows for a particle to be added at a specific location on the electric field. This state also allows for the quick addition of particles to a relative location using the mouse. A parameter is passed from the Main State to the Add Particle State when the state is entered. The Add Particle State uses this parameter to determine how the particle will be added, by mouse input or by keyboard input. This parameter is also used to determine whether the defaults of a negative or positive particle are to be put in the particle option pane.

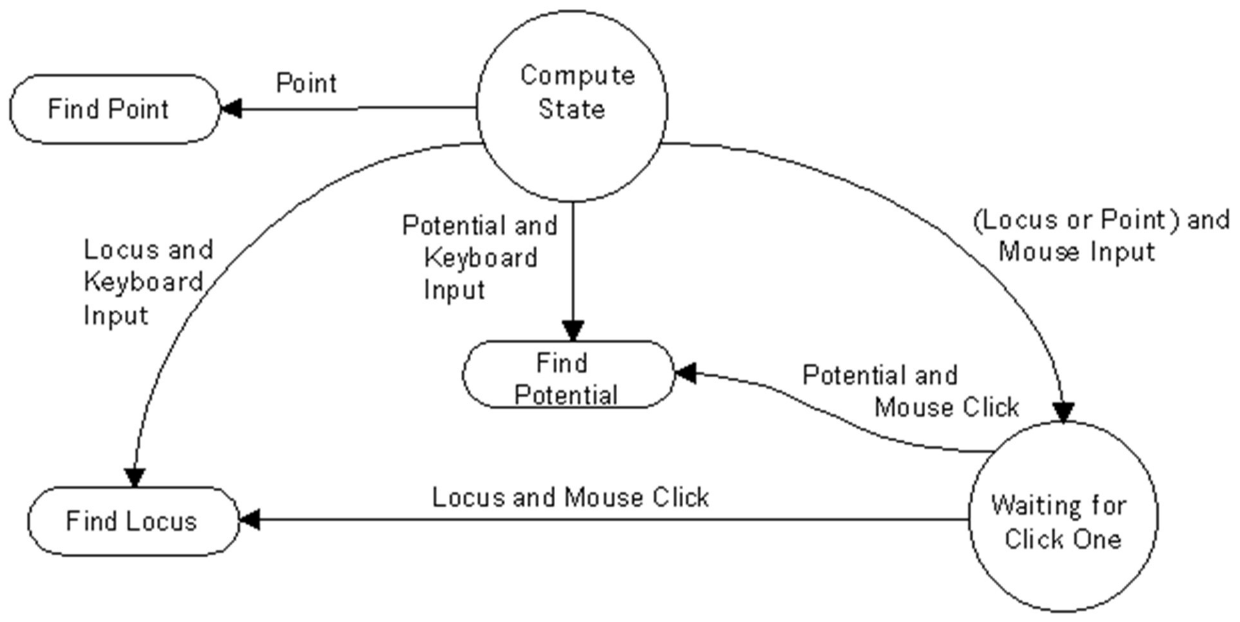
If the particle is to be added by mouse input, the Add Particle State moves in the Waiting for Click One Sub-State. This Sub-State is not exited until the Add Particle State is exited due to another action event occurring. When the mouse click action occurs, a particle is added to the electric field at the point clicked. If the particle is to be added by keyboard input, the Add Particle State gets the keyboard input from the user and adds the particle to the electric field.



The Configure Particle State handles changing the attributes of a particle. When a Canvas Right Action (right click) occurs, the Main State enters the Configure Particle State passing to this state the position where the Canvas Right Action occurred. The Configure Particle State then attempts to find a particle with this position. Upon finding a particle with this position, a Particle Menu (popup menu) is shown for the particle. The Configure Particle State then moves into the Waiting for Selection Sub-State until the user selects a menu item. Depending on the menu item selected, either a particle attribute is configured or the particle is removed from the field.



The Compute State handles computing the potential at a point, finding a point with a given potential, and finding equipotential surfaces. To find a point with a given potential, the state moves directly into this activity where it gets the potential to search for and then attempts to find it. Finding equipotential surfaces and computing the potential at a point can be accomplished quickly using the mouse. Since precision and accuracy are important in the study of equipotential surfaces, the point at which the potential is to be computed or an equipotential surface is to be found can be entered exactly using the keyboard. Similar to the Add Particle State, the Main State passes a parameter to the Compute State. This parameter is used to determine which computation activity is to take place and where the input for this activity is received (from mouse or keyboard).



If the input is to come from the keyboard, the Compute State moves directly into the activity that is to take place. If the input is to come from the mouse, the Compute State moves into the Waiting for Click One Sub-State. It stays here until a point on the field is clicked on. It then moves to the appropriate activity. This Sub-State is not exited until the Compute State is exited due to another action event occurring.

Algorithms

The following section discusses the algorithms used to find a point with a given potential, compute the potential at a point, and find an equipotential surface. These are all activities that are handled by the Compute State.

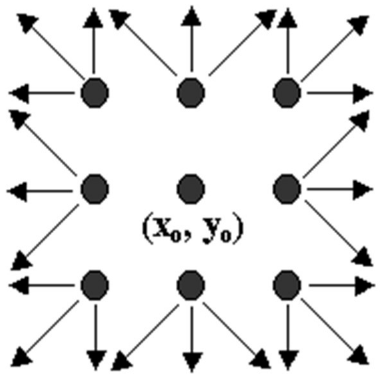
Compute the Potential at a Point

This is the simplest algorithm of the three discussed. To find the potential energy at a point, simply iterate through the particles on the field and plug in the values for q_n and r_n. The value of q_n is the charge field for the particle. Since r_n is the radial distance from the point in question to the nth particle, the distance formula can be substituted for r_n. The particle position field holds the location of the particle, say (x_n, y_n). If the point we are computing the potential at is (x, y), then r_n is the distance between (x, y) and (x_n, y_n). Thus, the new equation is:

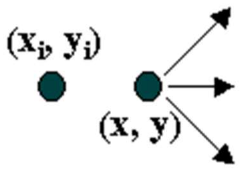
$$V = k\sum_{n} \frac{q_n}{\sqrt{\left((x-x_n)^2 + (y-y_n)^2\right)}}, \text{ where } r_n = \sqrt{\left((x-x_n)^2 + (y-y_n)^2\right)}$$

Find a Point with a Given Potential

This algorithm is more complex due to the nature of the equation. We are trying to solve for both x and y using the same equation given above. As the number of particles grows, this equation becomes more and more complex. Since the equation grows with the particles, it is nearly impossible to solve the equation for x or y, not to mention both. Thus, instead of blindly trying different values in hopes of finding the desired point, a search algorithm is used to find the point we are looking for. The first order of business is to select a point at which to begin the search, say (x_o, y_o). Since the field's points are points on a canvas, the algorithm can take advantage of the fact that each canvas point has exactly eight surrounding points. Of these surrounding eight points, the one with potential energy closest to the value being found is moved to. We then continue by searching the next three points in the same general direction until a point with a given potential is found. The point with potential energy closest to that of the value being searched for is moved to, and then, as before, the next three points in the direction we are moving are found. The figure below shows the next three points which are searched when going from (x_o, y_o) to a surrounding point.



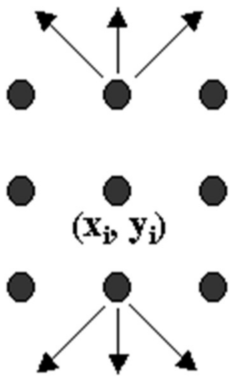
Looking at the above diagram, it seems as though for all of the eight cases the three points that are to be checked next would need to be specified. Thankfully, this is not so; the eight cases can be generalized into three cases. This generalization greatly enhances the running time of the algorithm. To explain the following three cases, let (x, y) be the point moved to and (x_i, y_i) be the previous point in the search (see example below).



Also let the difference in the x and y coordinates between these two points (x, y) and (x_i, y_i) be defined as follows:

$$\Delta x = x - x_i$$
$$\Delta y = y - y_i$$

The three cases for the algorithm can now be defined using D x and D y. The first case occurs when D x equals 0. This occurs when the search goes up or down along the y-axis. In this case, there is no horizontal difference between the points, only a vertical change. The diagram below shows a visual description of this case. The points with arrows emanating from them are the two (x, y) points that yield D x = 0. The arrows point to the next three points that should be searched for in this case.

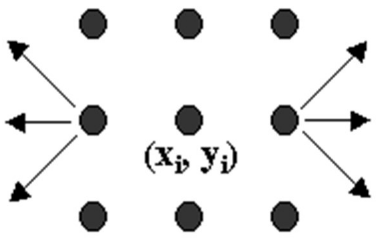


.

The three points the arrows point to can be given by the following:

$$(x-1, y+\Delta y)$$
$$(x, y+\Delta y)$$
$$(x+1, y+\Delta y)$$

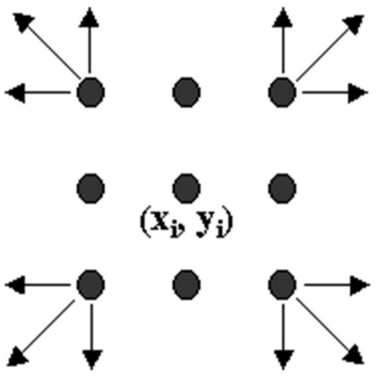
The next case for the algorithm occurs when D y = 0. This happens when the search moves left or right on the x-axis. Here there is no vertical change in the points, but a horizontal difference is present. The diagram below gives a visual representation of this case. Like before, the points with arrows emanating from them are the two (x, y) points that yield D y = 0. The arrows again point to the next three points that are to be searched for in this case.



The three points the arrows point to can be given by the following:

$$(x + \Delta x, y + 1)$$
$$(x + \Delta x, y)$$
$$(x + \Delta x, y - 1)$$

The final case encompasses all four of the corner points. This is the case where D x ^{1} 0 and D y ^{1} 0. In this case, there is both a horizontal and vertical difference between the points. The diagram below shows a visual representation of this case. The points with arrows emanating from them are the four (x, y) points that yield D x ^{1} 0 and D y ^{1} 0. The arrows again point to the next three points that are to be searched for in this case.



The three points that arrows point to can be given by the following:

$$(x + \Delta x, y + \Delta y)$$
$$(x, y + \Delta y)$$
$$(x + \Delta x, y)$$

The search for the point ends upon the occurrence of different events. One event that causes the end of the search is when a search point is not in the viewable field area. For example, if the canvas coordinates run from (0,0) to (100, 150), the search would end upon encountering a search point (x, y) such that x < 0 or x > 100, or y < 0 or y > 150. Upon this occurrence, a null pointer is returned since no point is found.

Another event that causes the end of the search is when the potential energy of the search point is significantly further from the potential energy being searched for than the potential found at the previous point. Let the given potential be p_o and the value of the potential energy at the search point be p_s. Define a value gamma (a) as the maximum difference between p_s and p_o.

When the difference between the points is greater than gamma, the search assumes we are not moving in the correct direction, and the search ends (see below). At this point, a null pointer is returned since no point is found.

$$|p_s - p_0| > \alpha \Rightarrow \text{point is null}$$

The third and optimal event that causes the end of the search is when a point that has the given potential is found. A margin of error epsilon (e) for the potential energy is allowed when searching for the point. If the difference between p_s and p_o is less than e, then the point with potential p_s is returned. This is the point with relative potential p_o.

$$|p_s - p_0| < \varepsilon \Rightarrow \text{point is found}$$

Find an Equipotential Surface

Finding an equipotential surface is another nontrivial matter. First a start point at or initial point (p_i) must be specified. Next, points with the same potential as this initial point must be found. To do this, we can make use of one of the properties of equipotential surfaces. This property tells us that all equipotential surfaces are continuous on the interval they encompass. From this, we know that the next point of the locus is directly next to the point we are at on the locus. (It is important to note that in the case of an "ideal" mathematical point, the potential has a singularity at that point. However, since the smallest known charged particle is an electron and electrons have a dimension, this "ideal" mathematical point does not exist in the real world.) By use of the before-mentioned property and the fact that we are dealing with canvas points disguised as real coordinates, we can look at the surrounding points to see which point gives us a potential equal to the point we are at.

After getting the initial point, the surrounding eight points are looked at to see which point to go to next. When the next point is found, it is added to the locus, and the search for more locus points is continued in the same direction. To do this, the same basic algorithm can be used for finding a point with a given potential.

The difference now is that when a point is found, it is added to the locus representing the equipotential surface. It then finds and moves to the next point and adds it to the locus until the entire equipotential surface is found. To know when to quit finding and adding points to the locus, another property of equipotential surfaces is used. This property tells us that an equipotential surface has a closed form. Since we know the starting or initial point (p_i) of the locus, we simply keep finding and adding points until the point we find (p) is equal to p_i.

$$p_i = p \Rightarrow \text{equipotent ial surface found}$$