Topics: Visualizing Var::N, Var::L, Var::d
How to simulate a charge layer
Sample graphic of a disk-shaped charge layer

⊗ Simulation Domain: Var::L.init(30e+3, 30e+3, 70e+3)

($L_x = 30$ km, $L_y = 30$ km, $L_z = 70$ km)

Conceptually, Var::L tells us the size of the simulation domain, in *meters*. Think of "L" as the *length* of a dimension.

Above, we saw the line of code that initializes the simulation domain, "Var::L.init()". Broken down, this correlates to:

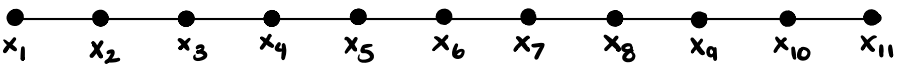
- Var::L.x = L_x = "x-domain" $\in [0, 30E+3] = [0 \text{ meters}, 30 \text{ kilometers}]$
- Var::L.y = L_y = "y-domain" $\in [0, 30E+3] = [0 \text{ meters}, 30 \text{ kilometers}]$
- Var::L.z = L_z = "z-domain" $\in [0, 70E+3] = [0 \text{ meters}, 70 \text{ kilometers}]$

⊛ Number of Grid Points: Var::N.init(31, 31, 71)

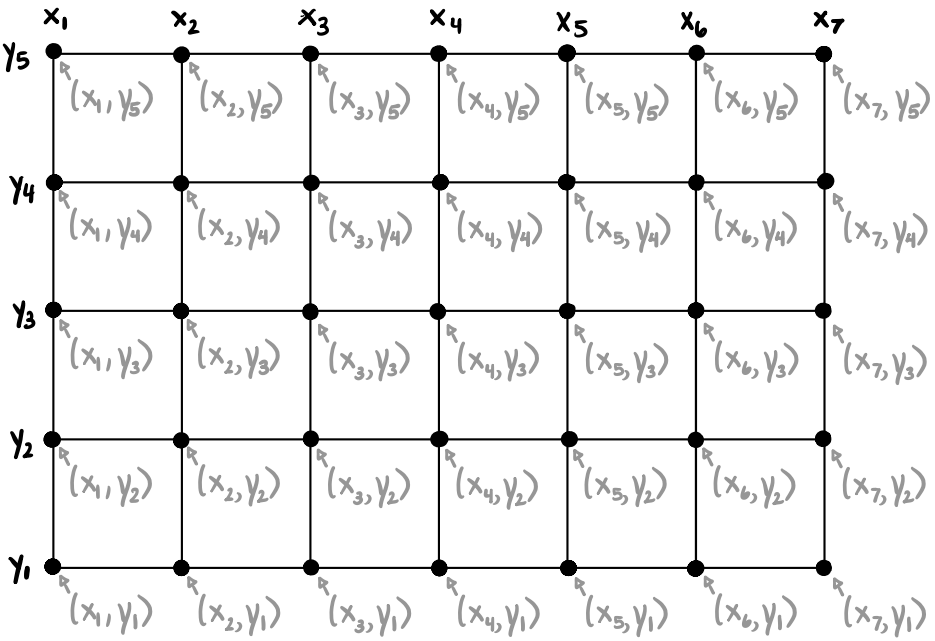
(N_x, N_y, N_z)

Since we are creating a numerical simulation, we use a *grid* to replicate a multi-dimensional space. Calculations are performed between these grid points to replicate real life in a discrete fashion. Below are some representations of 1D and 2D grids.

- $N_x = 11$:



- $N_x = 7$, $N_y = 5$:



Above, we saw the line of code that initializes the simulation domain, "Var::N.init()". Broken down, this correlates to:

- Var::N.x = N_x = "# of grid points in x-domain" = 31
- Var::N.y = N_y = "# of grid points in y-domain" = 31
- Var::N.z = N_z = "# of grid points in z-domain" = 71

⊛ Separation between Grid Points: Var::d.init(Var::L, Var::N)

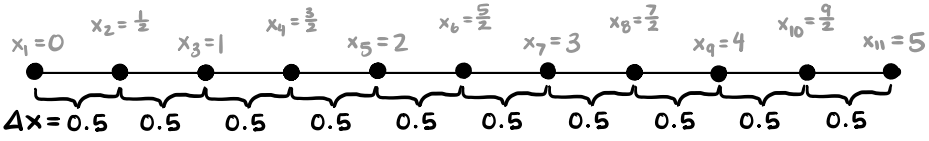
We've covered the representations of the multi-dimensional space (Var::N) as well as the simulation domain (Var::L), but how do we tie it together? This is where Var::d comes in. Var::d is the quantified *spacings* between grid points. Think of "d" as the *difference* "Δ" between grid points in a spacial domain.

In the file "src/ResGrid.cpp", we see that the corresponding equation to define the values of Var::d are:

$$\Delta x = \frac{L_x}{N_x - 1}, \quad \Delta y = \frac{L_y}{N_y - 1}, \quad \text{and} \quad \Delta z = \frac{L_z}{N_z - 1}$$

As an example:

- $L_x = 5, N_x = 11, \Delta x = \frac{5}{11-1} = \frac{1}{2}$:



Above, we saw the line of code that initializes the simulation domain, "Var::d.init()". Broken down, this correlates to:

- Var::d.x = Δx = "spacing between x-domain grid points" = $\frac{30\text{ km}}{31-1}$ = 1 km = 1,000 m
- Var::d.y = Δy = "spacing between y-domain grid points" = $\frac{30\text{ km}}{31-1}$ = 1 km = 1,000 m
- Var::d.z = Δz = "spacing between z-domain grid points" = $\frac{70\text{ km}}{71-1}$ = 1 km = 1,000 m

⊛ Spacial Shifts: Var::z_gnd; Var::z_shift; Var::y_shift

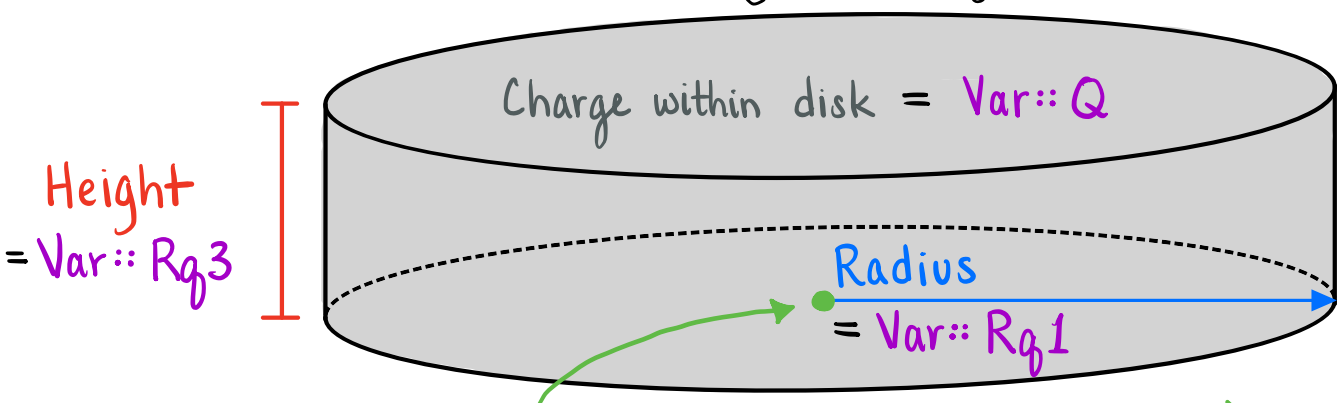
- Var::z_gnd: Ground elevation, depends on surface topography (Eg: "mile-high city" of Denver, Colorado, z_gnd = 1610 m).
- Var::z_shift: Vertical displacement of the cloud, automatically based on ground elevation (i.e. z_shift = z_gnd).

- Var::y_shift: Horizontal influence of a windshear; relevant when Var::LoadingType == CURRENTS (typically, we will have the loading type as SET_CHARGES for beginning cases).

⊛ Shape of a Charge Layer: Var::C.disk(...)

↑ "cloud" or "charge layer"

While there are several shapes of charge layers we can simulate, we will be primarily focusing on *disks*:



For most charge layers, the values of Var::Xq and Var::Yq are assumed to be at the center of the simulation domain, i.e.:

$$Var::X_q = Var::L.x/2 \quad ; \quad Var::Y_q = Var::L.y/2$$

The height at which the charge layer (cloud) begins however, is dependent on *two* user inputs. The first is the ground elevation (Var::z_gnd = Var::z_shift), and the other is the distance above the ground:

$$Var::Z_q = (\text{distance above the ground}) + Var::z\_shift$$

That takes care of the *positioning* of the cloud. Next we have to take care of the *dimensions* of the cloud. The disk's *radius* is Var::Rq1 while the *height* is Var::Rq3.

Finally, we have the *charge* contained in the cloud, Var::Q. Combining this information and some simulation parameters, you can simulate a disk shaped charge layer with the following:

Var::C.disk(① Var::Q, ② Var::Xq, ③ Var::Yq, ④ Var::Zq, ⑤ Var::Rq1, ⑥ Var::Rq3, ⑦ Var::d, ⑧ Var::N)

⊛ Simulating a Charge Layer: Var::LoadingType = SET_CHARGES

- Step 1: Define the parameters...

① Var::Q = (#); In Coulombs
② Var::Xq = Var::L.x/2; In meters, automatically calculated
③ Var::Yq = Var::L.y/2; In meters, automatically calculated
④ Var::Zq = (#) + Var::z_shift; In meters
⑤ Var::Rq1 = (#); In meters
⑥ Var::Rq3 = (#); In meters

- Step 2: Initialize the charge layer...

Var::C.disk(①, ②, ③, ④, ⑤, ⑥, Var::d, Var::N);

- Step 3: Include the charge layer in the simulation...

Var::ChargeCfg.push_back(Var::C);

⊛ Example:

Say we have a disk shaped cloud that has a uniform charge density of $1\ \frac{nC}{m^3}$. The cloud is 3 km thick and 14 km wide. The station that observed the cloud is 3km above sea level, and the cloud was 37 km above the station. Assume we know the particle number density at intervals of 1 km up through 70 km, and we want to simulate a square ground region that is at least 15 km around the station.

- Step 1: Define the parameters, starting with the easiest ones:

radius: $Var::Rq1 = \frac{14\ km}{2} = 7\ km$
height: $Var::Rq3 = 3km$
charge: $Var::Q = (1\times10^{-9}\frac{C}{m^3})[\pi\cdot(7\times10^3\ m)^2\cdot(3\times10^3 m)] \approx 462$ Coulombs
z_0: $Var::z\_gnd = Var::z\_shift = 3\ km$
domain:
$Var::L.x = (2\cdot 15\ km) = 30\ km$
$Var::L.y = (2\cdot 15\ km) = 30\ km$
$Var::L.z = 70\ km$
grid points:
$Var::N.x = (Var::L.x\cdot\Delta x)+1 = 31$
$Var::N.y = (Var::L.y\cdot\Delta y)+1 = 31$
$Var::N.z = (Var::L.z\cdot\Delta z)+1 = 71$
center:
$Var::Xq = Var::L.x/2 = 15\ km$
$Var::Yq = Var::L.y/2 = 15\ km$
$Var::Zq = 37\ km + z\_shift = 40\ km$

Combining the information above & Steps 2-3 lead us to the following lines of code:

Top of script:

```
Var :: N.init(31,31,71);
Var :: L.init(30e+3,30e+3,70e+3);
Var :: z_gnd = 3.0e+3;
```

Under Var::isNewRun = SET-CHARGES section:

```
Var :: Q = 462;  Var :: Xq = Var :: L.x/2;  Var :: Yq = Var :: L.y/2;
Var :: Zq = 37e+3 + Var :: z_shift;  Var :: Rq1 = 7.0e+3;  Var :: Rq3 = 3.0e+3;
Var :: C.disk(Var :: Q, Var :: Xq, Var :: Yq, Var :: Zq, Var :: Rq1, Var :: Rq3, Var :: d, Var :: N);
Var :: ChargeCfg.push_back(Var :: C);
```

Graphical Depiction of Result: