Run Time Interactive Graphics in Particle-in-Cell codes

Viktor K. Decyk
UCLA

Interactive graphics based on Python: using tkinter, numpy, and matplotlib

Tkinter is a graphical user interface (GUI) included with Python

Model-View-Controller (MVC) design pattern, in multi-threaded shared memory system

- MVC establishes a separation of concerns
- Controller and graphics runs in main thread, physics runs in other thread(s)
- Controller is event driven, and it controls View, and synchronizes with physics
- Only View imports matplotlib, does not generally process events
- Physics may use OpenMP in Fortran or C dynamic libraries, does not use tkinter or matplotlib

Separate Python scripts cannot safely share global memory

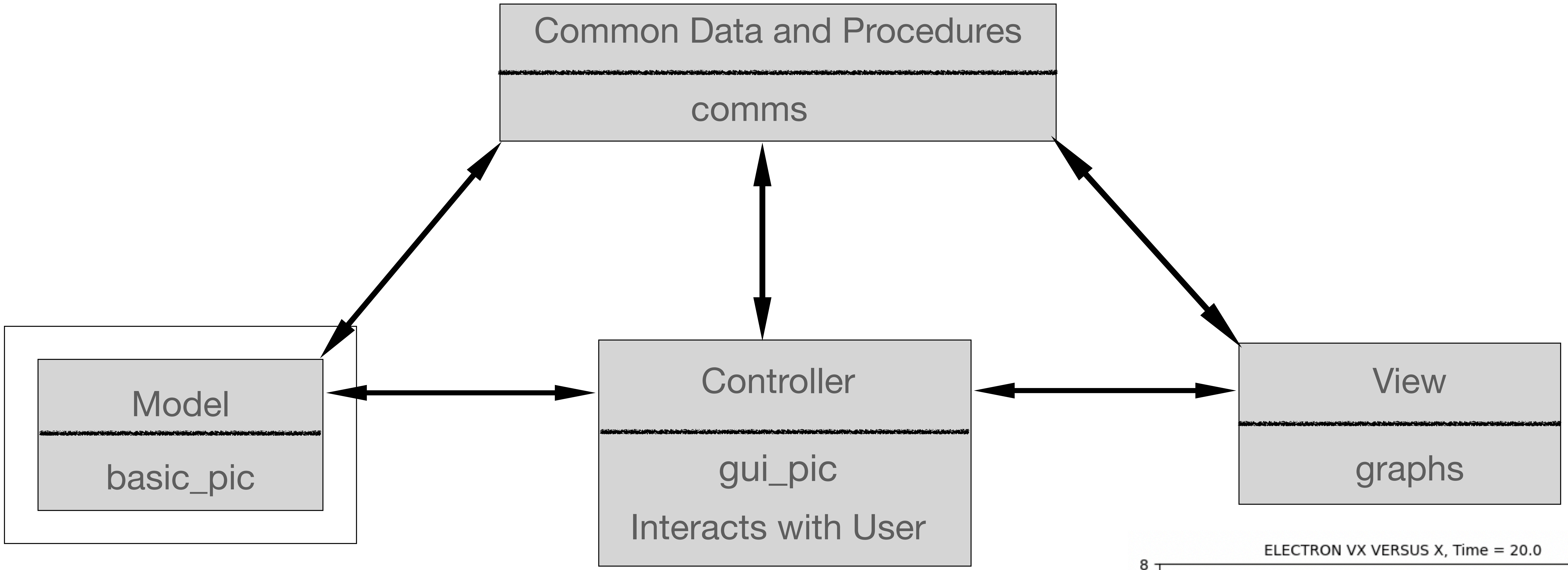
- But they can share a common global memory

Only two ways to safely communicate between threads

- By generating events
- By writing to thread-safe queue objects

Alan D. Moore, "Python GUI Programming with Tkinter", 2021

Model-View-Controller Design Pattern



Common Data and Procedures

comms

Model

basic_pic

Controller

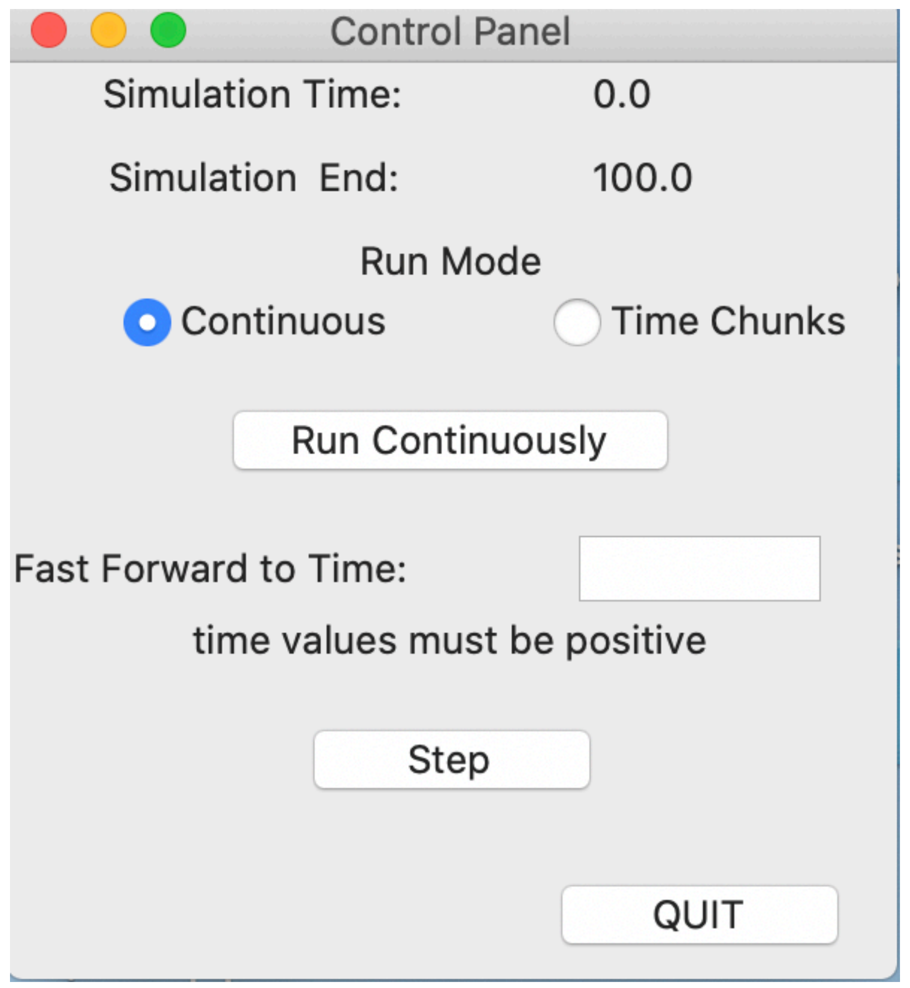
gui_pic

Interacts with User

View

graphs

Physics runs in separate thread(s)



Control Panel

Simulation Time: 0.0

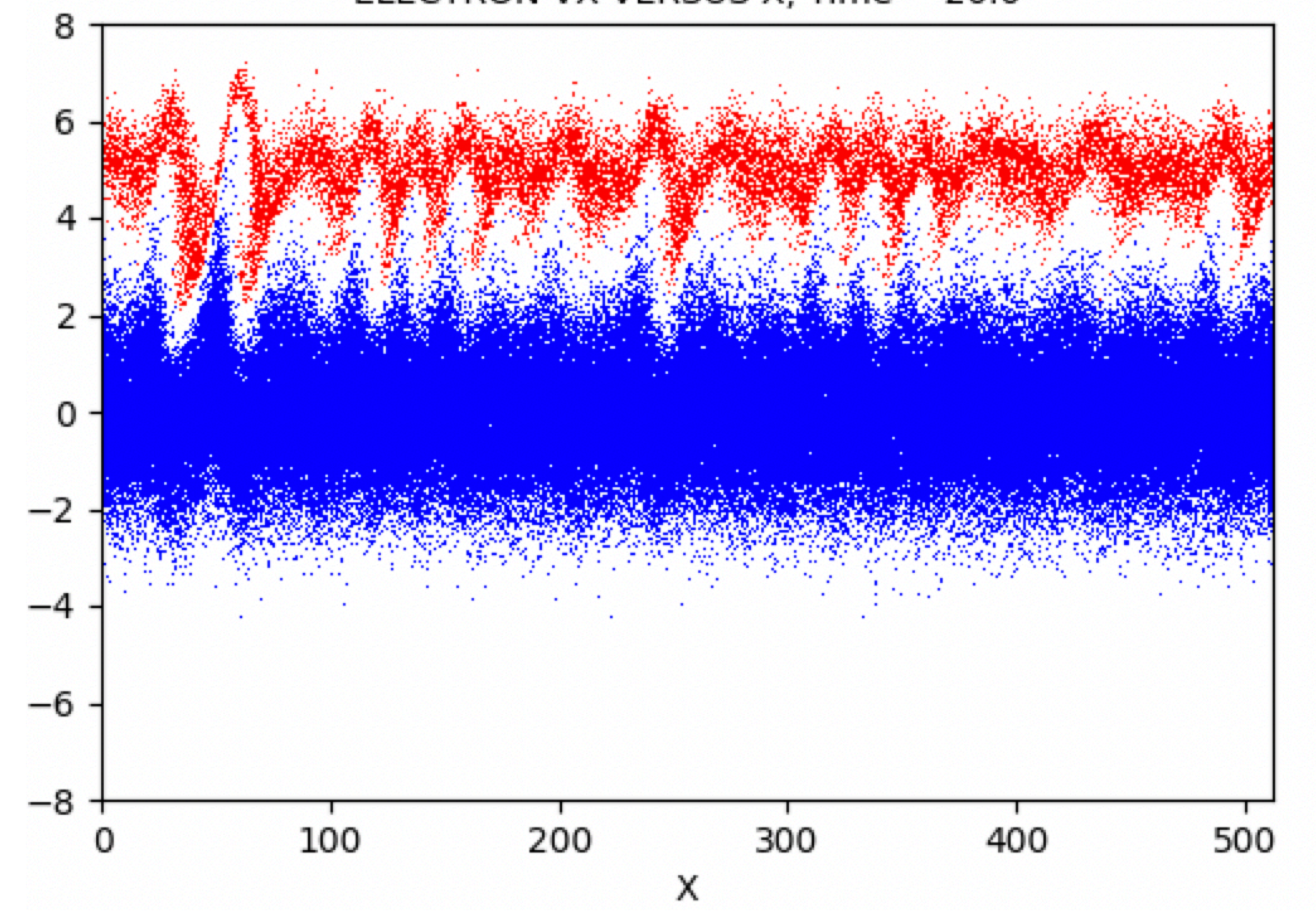
Simulation End: 100.0

Run Mode

● Continuous ○ Time Chunks

Fast Forward to Time:

time values must be positive



ELECTRON VX VERSUS X, Time = 20.0

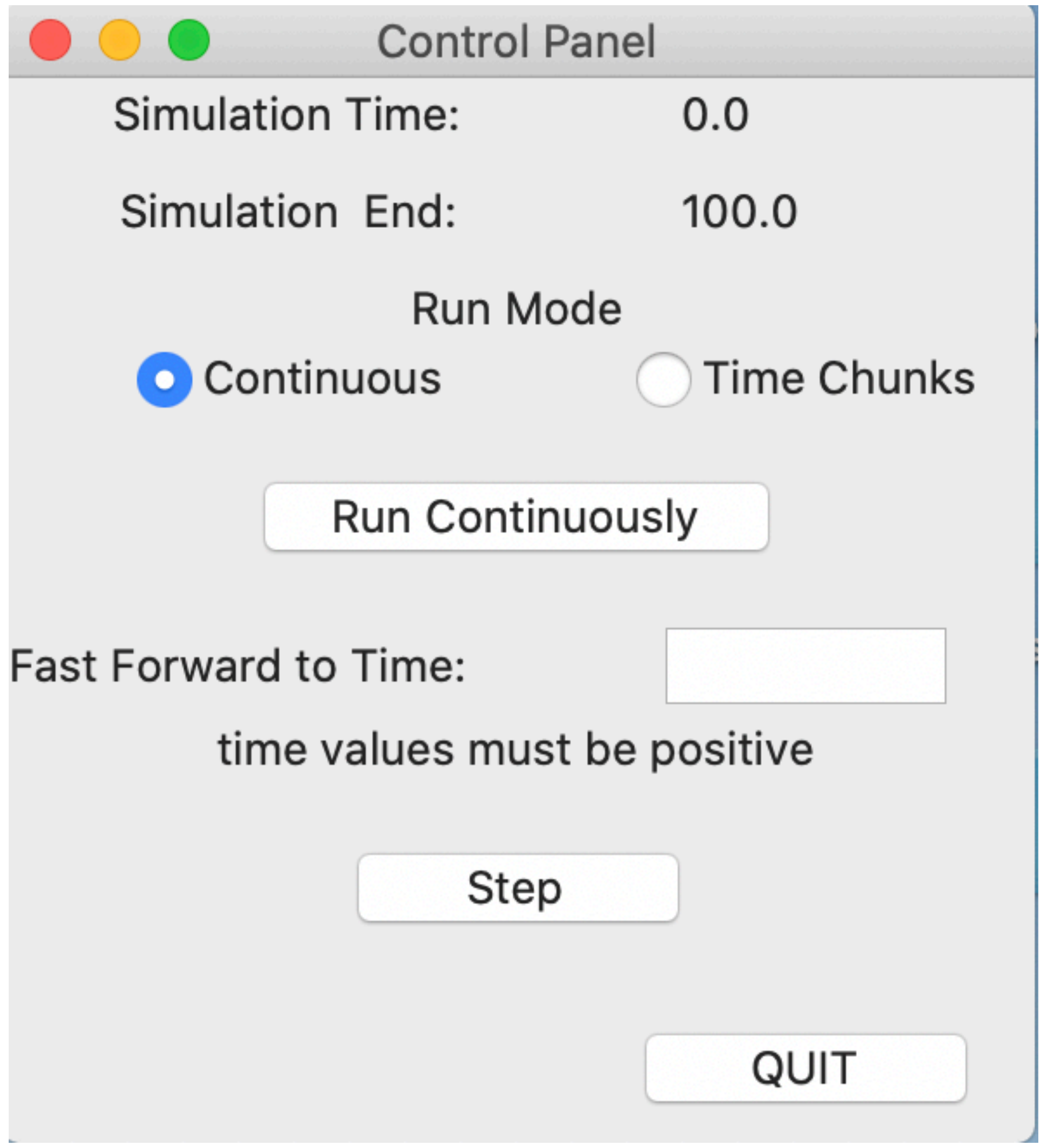
Demonstration

Execute: python3 gui_pic1.py
you should see the control panel and four plots in a window

The current simulation time and end time are shown
Step: Goes one time step and pauses
Run Continuously: Runs and plots without pausing

Since physics often runs faster than plots:
Fast Forward Time: Runs to designated time without plotting
Run Chunk: Runs without plotting to current + jump time

Quit: Terminate Program



Control Panel

Simulation Time: 0.0

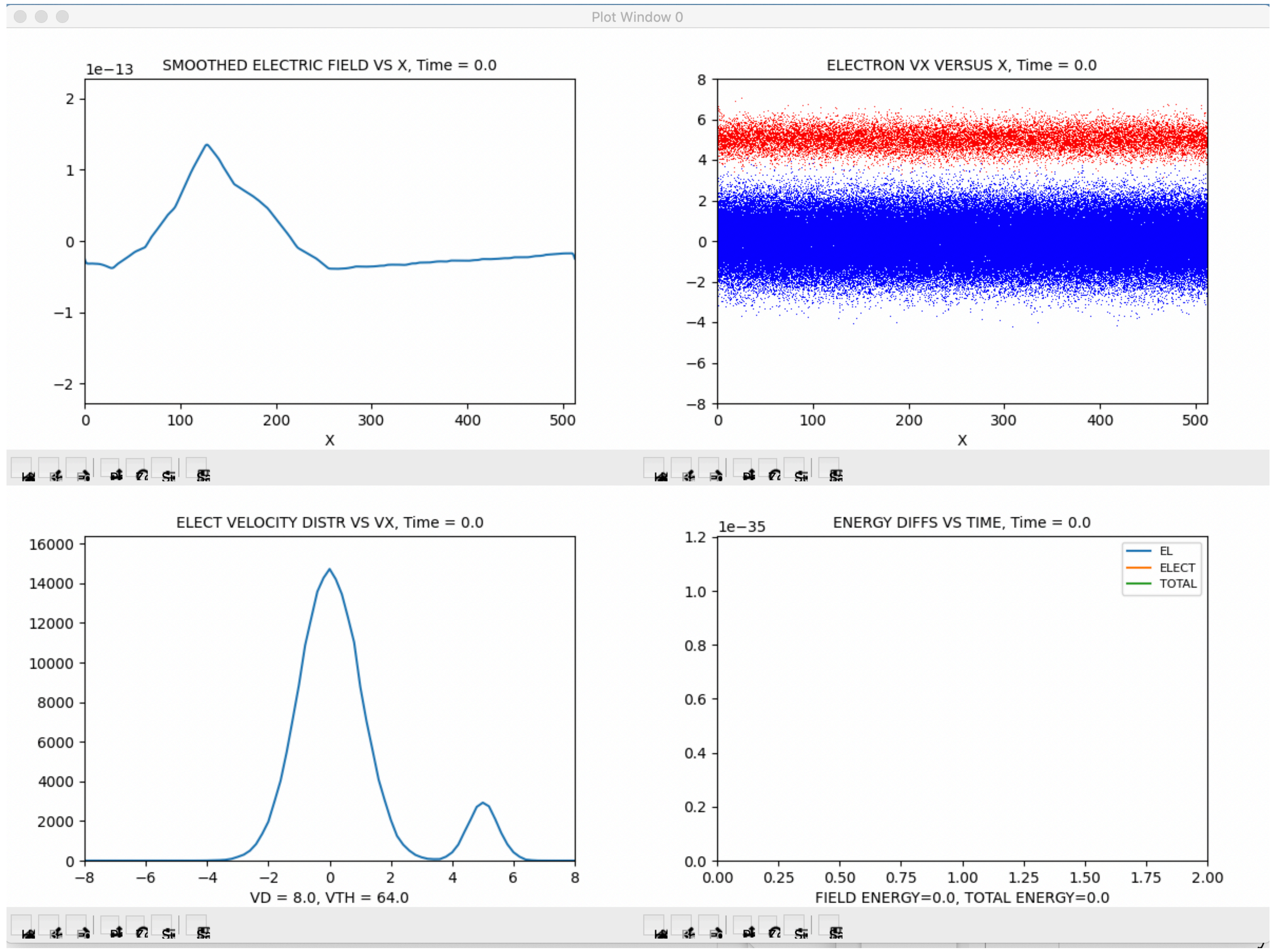
Simulation End: 100.0

Run Mode

● Continuous ○ Time Chunks

Fast Forward to Time:

time values must be positive



Graphs

The module Basic_Gui/graphs.py contains the matplotlib procedures currently supported:

- dscaler1: displays 1d scalar field vs x
- displayfv1: displays 1d velocity distribution function vs. vx
- grasp1: displays 1d x-vx particle phase space x vs. vx
- displayw1: displays time history of kinetic, field, and total energies relative to initial values

You can add your own procedures to this library

- each call to your procedure should have a unique label
- each procedure should end with comms.set_plot_status(label)

The labels are used by controller gui_pic1.py to call the display procedures when responding to an on_plotstart event requested by the physics code basic_pic1.py

Adding new plots in the basic_pic1 program

Just before main iteration loop, the procedure comms.update_gui sends two python dictionaries
The first contains of plot labels and indices. You should add your label and the next index.

- {"Label for first plot":0,"Label for second plot":1,...}

The second contains of names of constants and values needed by the plots.

- {"DT": 0.1,"TEND": tend,...}

You should add the constants and values needed by your plot.

For each plot add your unique label and the arrays needed for your display:

- **comms.update_plot(label,plotdata,plotdata2**)

If you require more than two arrays, you can modify the comms.update_plot function,

- alternatively, copy extra array to a new global constant you create in comms.py

Then wait for your plot to complete by adding:

```
gui_err = comms.check_plot_status()
if gui_err != comms.plot_name:
   if (gui_err=='QUIT'): break
```

Adding new plots in gui_pic1.py

Finally, in gui_pic1.py, add your plot to the if-then else block in the function on_plotstart:

```
elif plot_name=='Your unique label':
   graphs.your_new_plot(comms.plot_data,plot_name,comms.plot_data2,
                                   your new constants from dictionary)
```