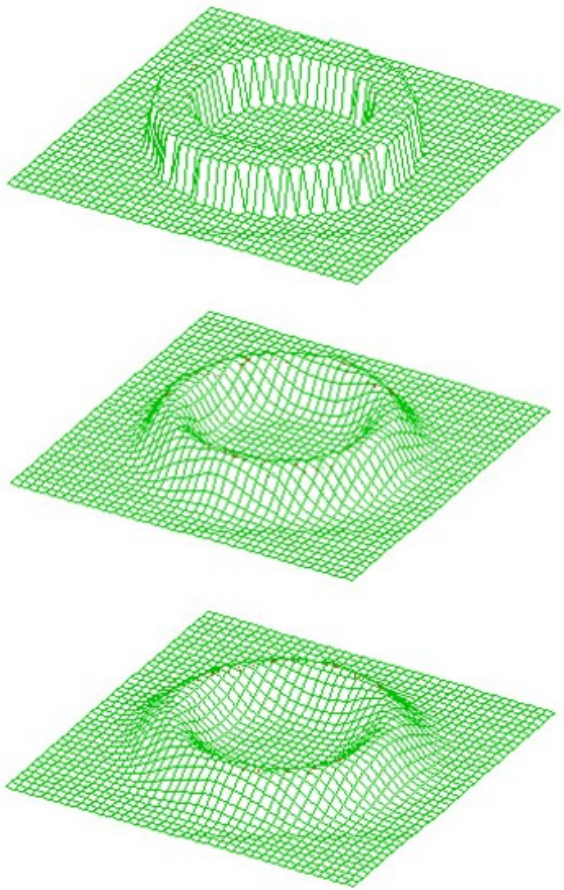


Real time display with *Gnuplot*

Ge Baolai, Western University
SHARCNET | Compute Ontario
Digital Research Alliance of Canada

Outline

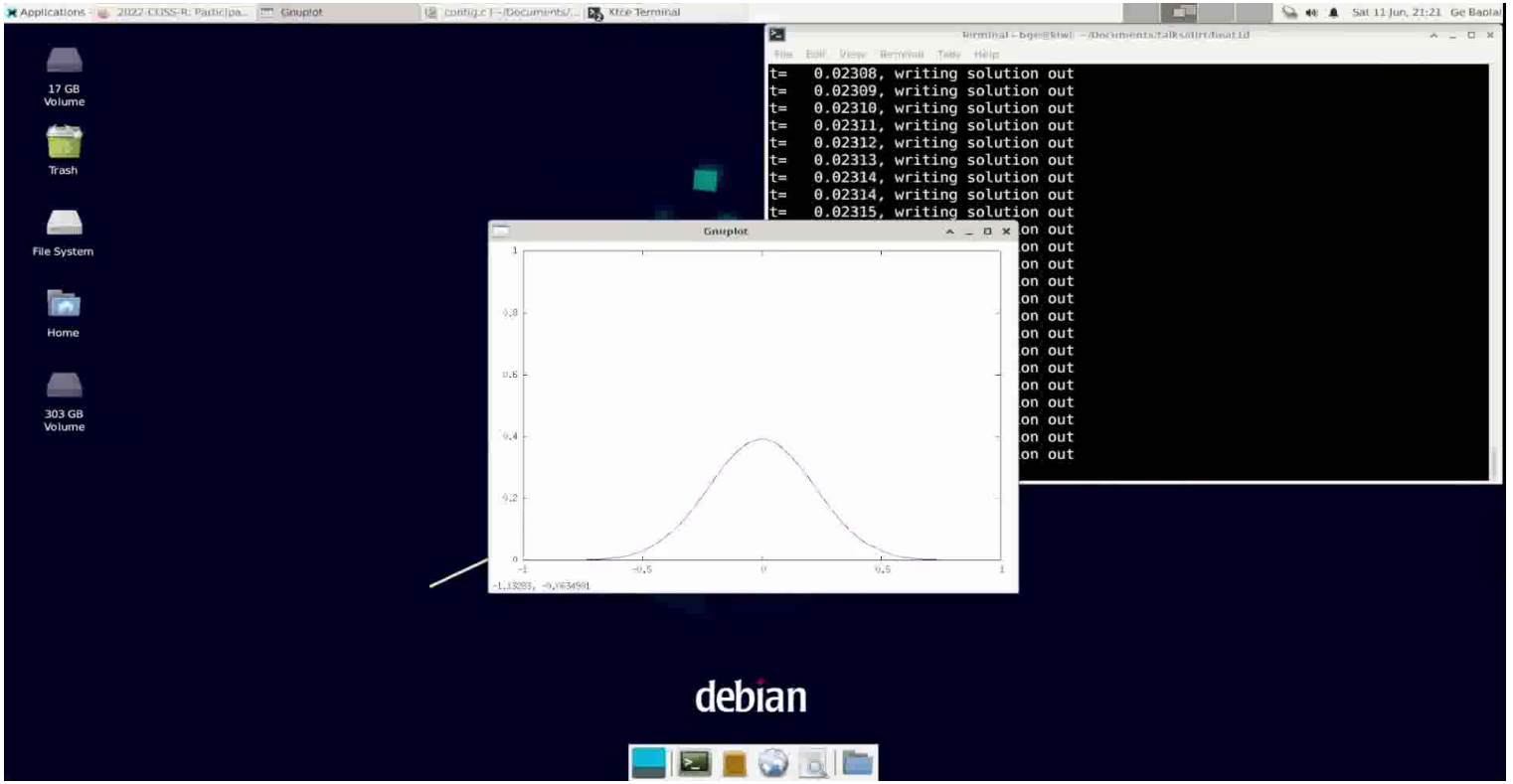
- The needs – compute and visualize
- The workflow and requirements real time display
- Computing and visualizing – calling third party software directly from the application
- Computing and visualizing – separate, asynchronous processes
- Computing and visualizing – synchronized processes via a pipe
- Other options

The needs

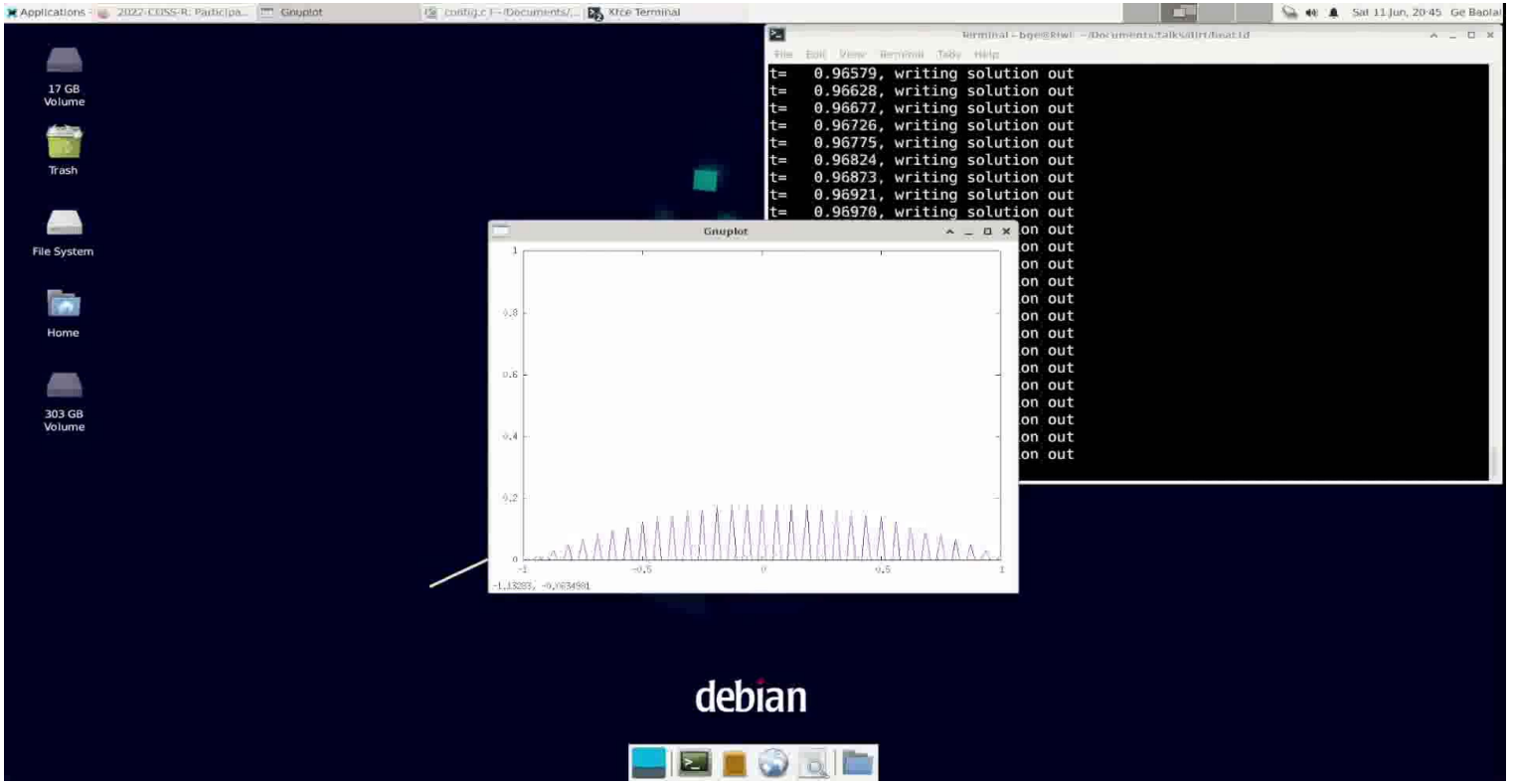
The needs

- Experimenting an algorithm, a configuration.
- Needs to check the intermediate results, stop the computation if things go unexpectedly.
- Debugging by visualization, especially useful for physics and mathematical problems.

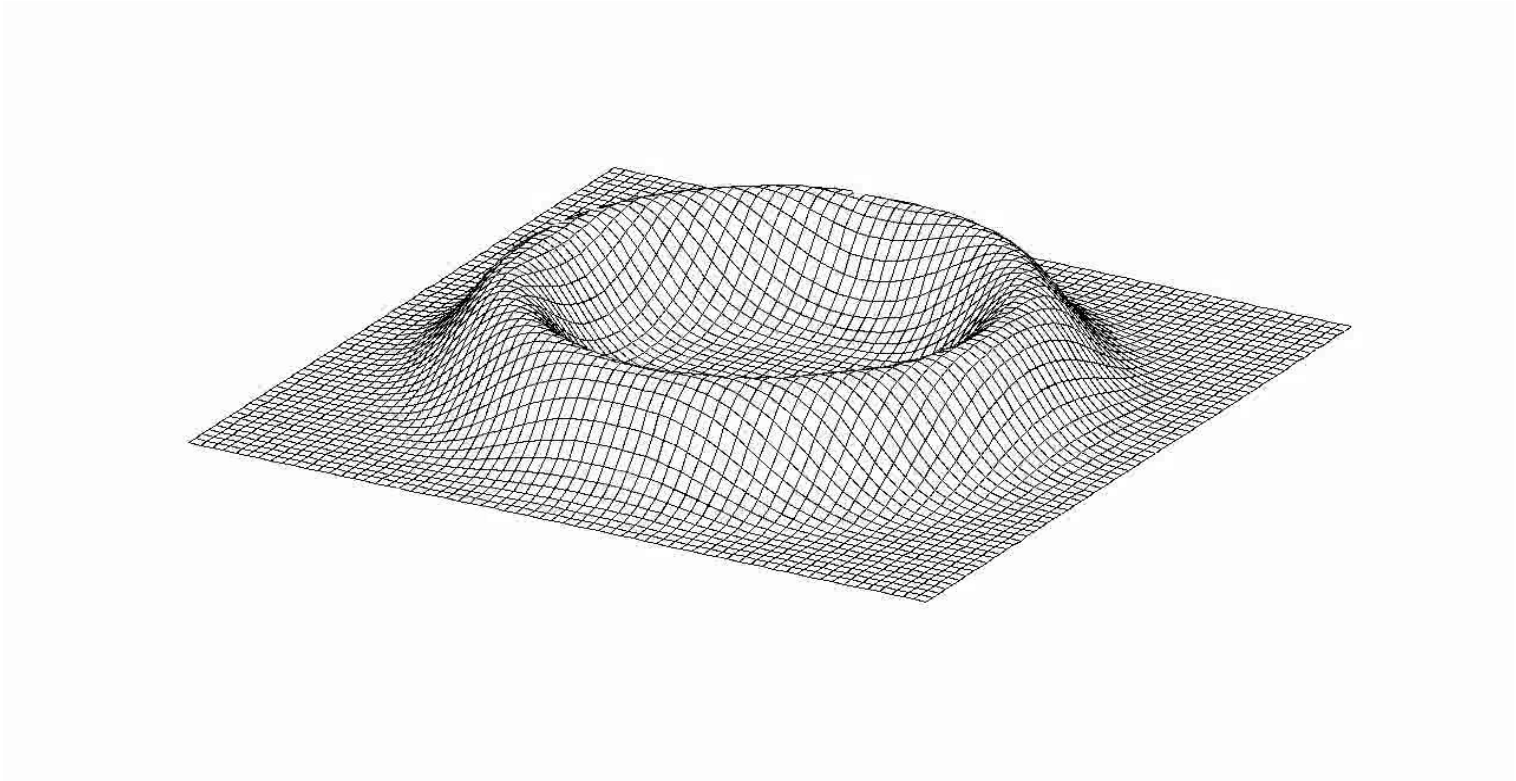
The needs – checking solutions



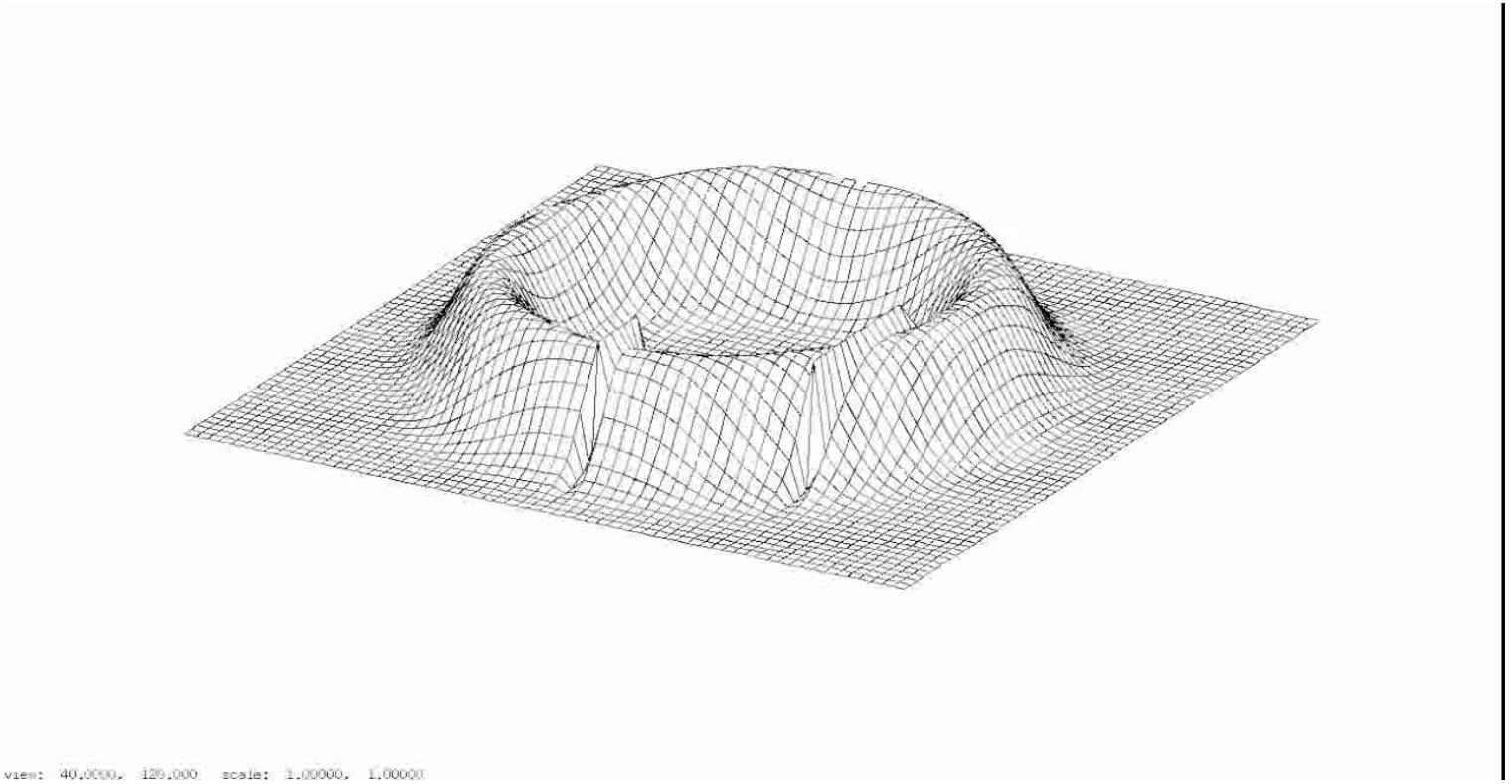
The needs – checking solutions



The needs – visual debugging



The needs – visual debugging



The workflow and requirements for real time display

The workflow and requirements - methods

Push

- The graphical program (e.g. Gnuplot) is fed with data stream from the application program.
- It generates the plot synchronously.
- Calling a third party tool directly, e.g. PGPLOT from within the application program belongs to this category.
- Other options include, e.g. using a pipe that is managed by the operating system.

Pull

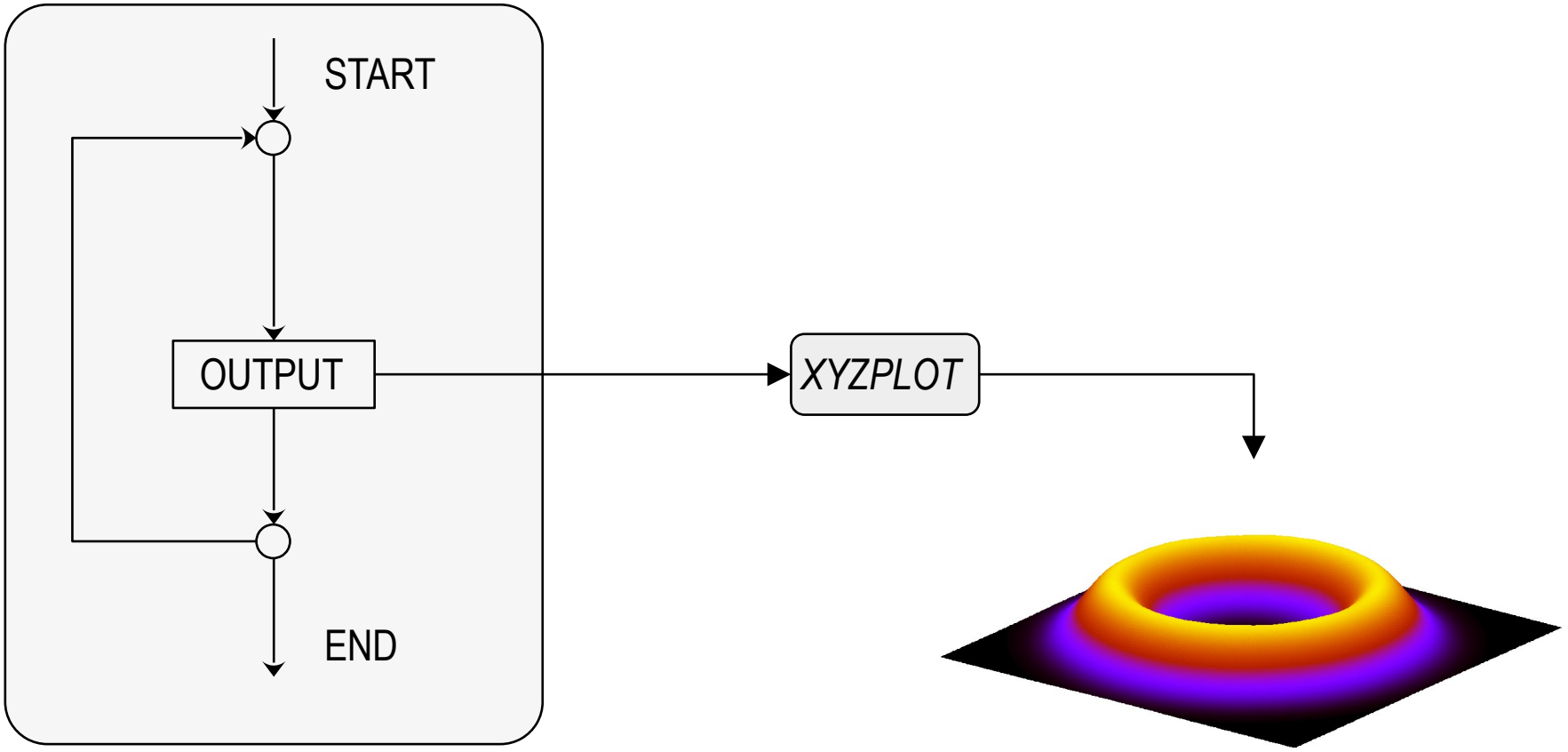
- The graphical program does not know when the data arrives.
- It checks periodically if the data file has become available.
- When the data is ready, it generates the plot.
- This is the common way.

The workflow and requirements - expectations

- The display is platform independent.
- The display of the output is smooth.
- No missing “frames” or corrupted graphs.
- Easy to implement.
- Do not slow down the compute process.

Computing and visualizing via calls to third party graphics functions

Compute & display – synchronously via 3rd party functions



Compute & display – synchronously via 3rd party functions

Using PGPLOT

```
! Initialize PGPLOT
call pgbeg(0, "/xs", 1, 1)
call pgask(0)
call pgenv(xlim(1), xlim(2), 0., 1.0,  0,  0)
call pgline(n,x(1),uold(1))

do step = 1, num_steps
  ! Compute the solution
  do concurrent (i=1:n)
    unew(i) = (1.0 - 2*r)*uold(i) + r*(uold(i-1) + uold(i+1))
  end do

  ! Output the current solution
  if (mod(step, output_steps) == 0) then
    call pgeras
    call pgline(n,x(1),unew(1))
  endif

  ! Swap the storage - swap pointers instead of arrays
enddo
```

- PLplot
- pyplot-fortran
- f03gl
- Gnuplot-iostream-interface
- MathGL
- … ...

Compute & display – synchronously via 3rd party functions

The workflow

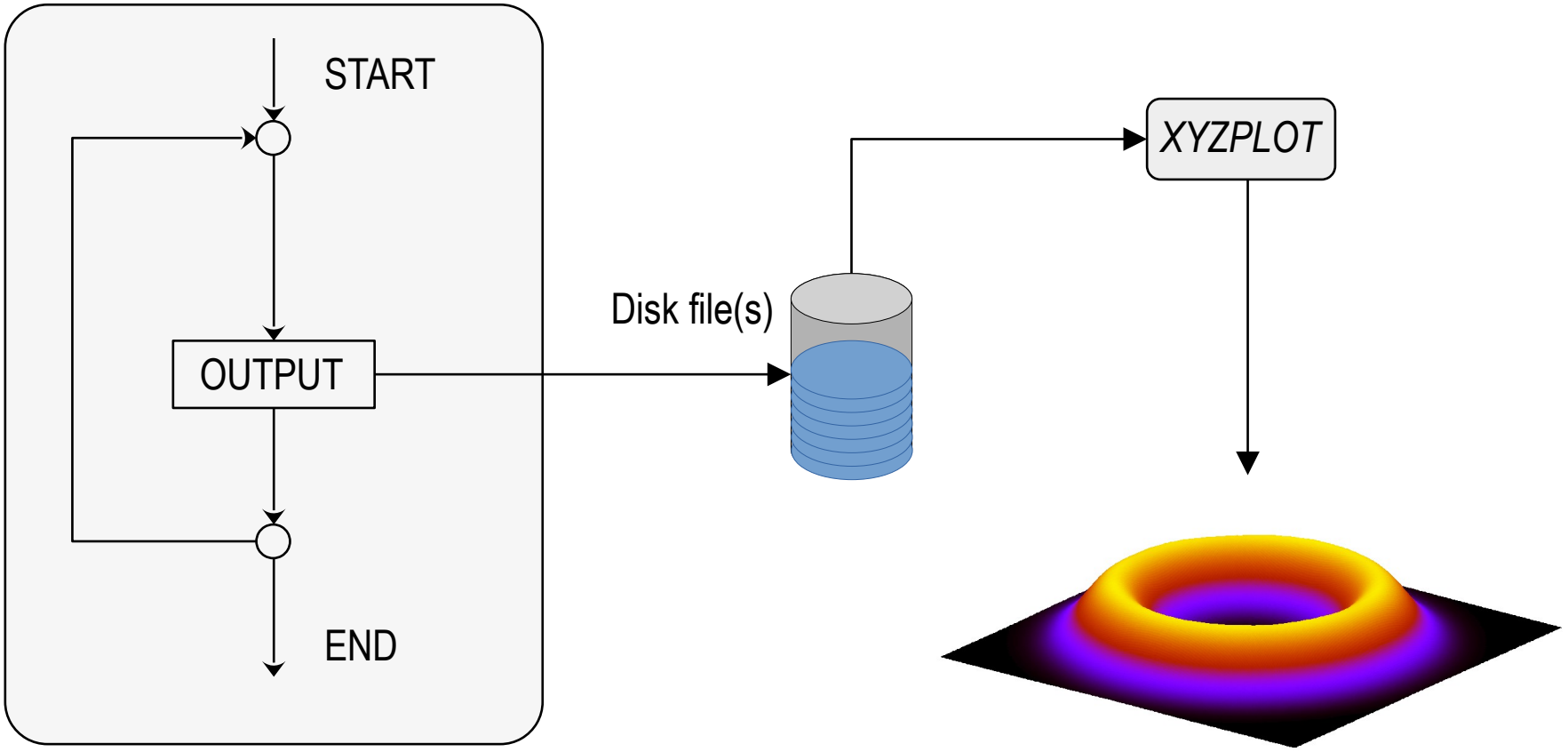
- The compute process produces an output and calls a third party routine, e.g. PGPLOT, to produce (persistent) display.
- **Pros:** Synchronized.
- **Cons:** The graphical display usually tied to the language. Not portable, not so easy to customize.

Meet the requirements?

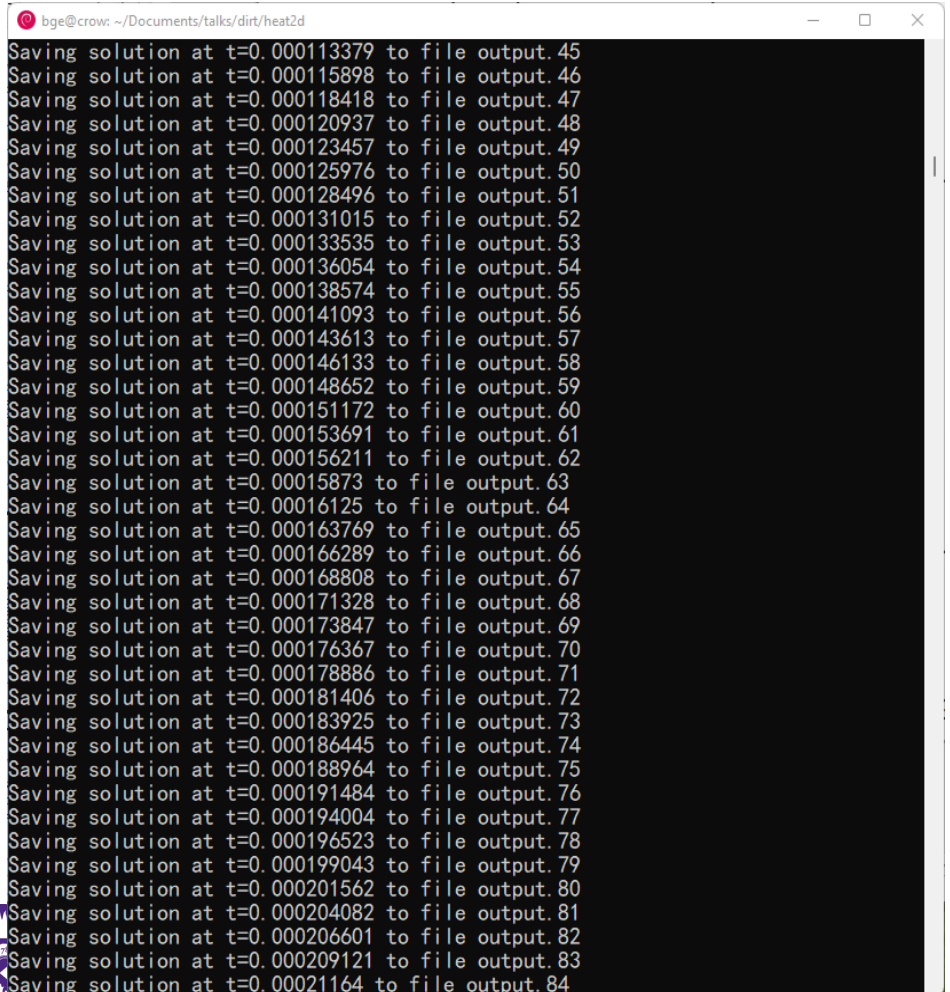
- The graphical routine call blocks, is able to read the output from the main with no loss or corrupted data, until the plot is generated.
- To draw (frame of) plot within the same figure window, making animation in real time possible.

Computing and visualizing via separate, asynchronous processes

Compute & display – asynchronously



Compute & display – asynchronously



Gnuplot script

```
# “heat2d_mplots.plt” – Generate multiple plots in a same graph
set view 30,104,1,1
unset colorbox
set hidden3d

num_output = ARG1 # Version 5+, taking ARG1 to ARG9

do for[i=1:num_output] {
   splot 'output.'.i using 1:2:3 with lines lt -1
   pause -1
}
pause -1 "Press ENTER to exit"
```

The Shell command

```
gnuplot -c heat2d_mplots.plt 200
```

demo

Compute & display – asynchronously

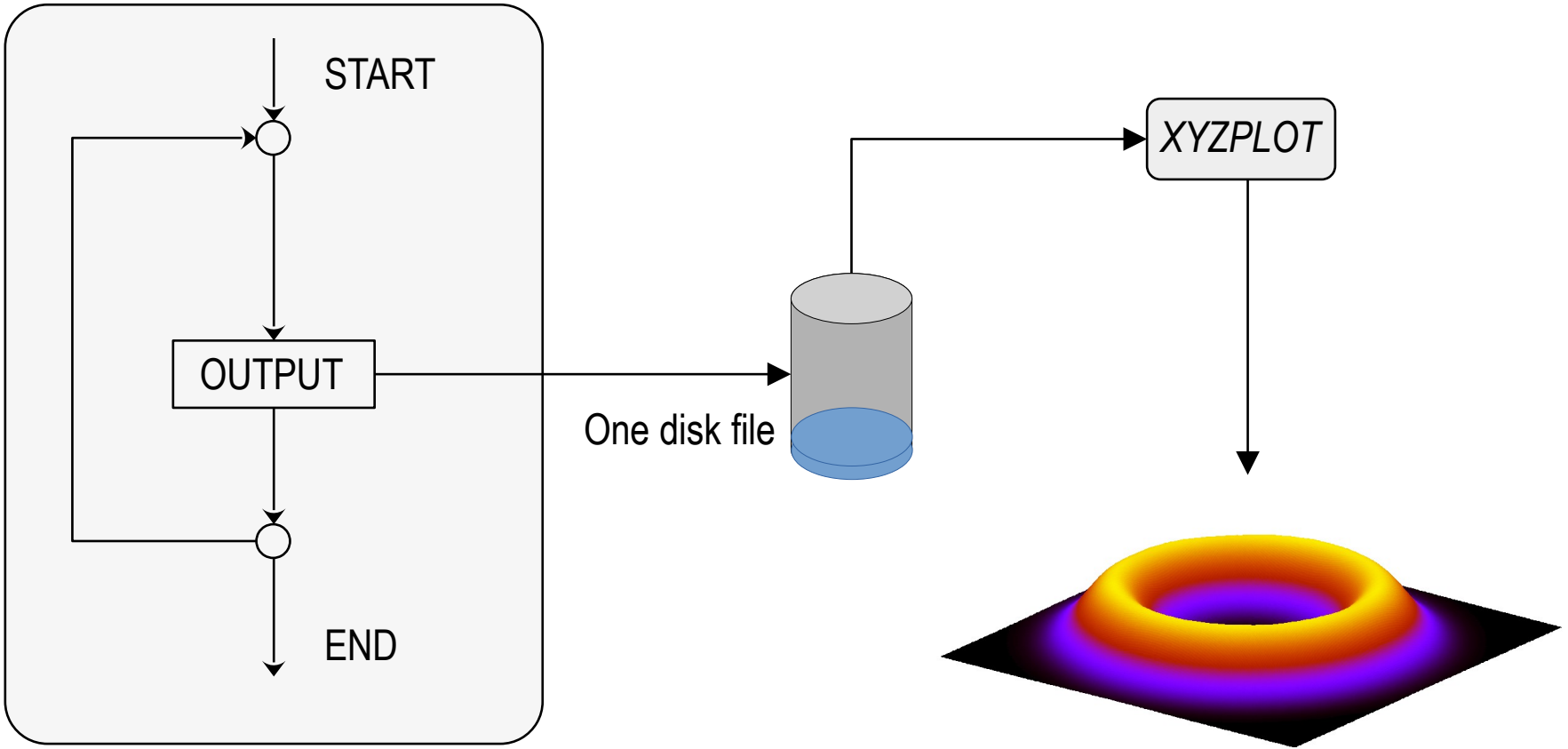
The workflow

- This is the most common approach.
- The compute process produces snapshots of output periodically and stores them into a sequence of files, which can be processed asynchronously and examined in post-processing.
- We use a separate process to view the results in sequence with some delay.
- **Pros:** No change in the compute program, the program knows nothing about graphical display.
- **Cons:** Many many files need to be saved in order to show a relatively long computed solution.

Meet the requirements?

- The graphical program reads data for each frame from a separate file, intact, so no loss.
- The graphical program can generate multiple frames within the same figure window, make animation in real time possible.

Compute & display – asynchronously



Compute & display – asynchronously

The application code

```
// During the iteration, when its time to output

   if (NULL == (fp = fopen("output.dat", "w")))
   {
      printf("Can't open file %s: %s", fname, strerror(errno));
      exit(0);
   }

   printf("Saving solution at t=%g to file %s\n", t, fname);
   for (int j = 1; j <= nyg; j++)
   {
      for (int i = 1; i <= nxg; i++)
         fprintf(fp, "%f %f %f\n", xc[i], yc[j], unew[j][i]);
      fprintf(fp,"\n");
   }

   icount++;
   fclose(fp);
```

Gnuplot script

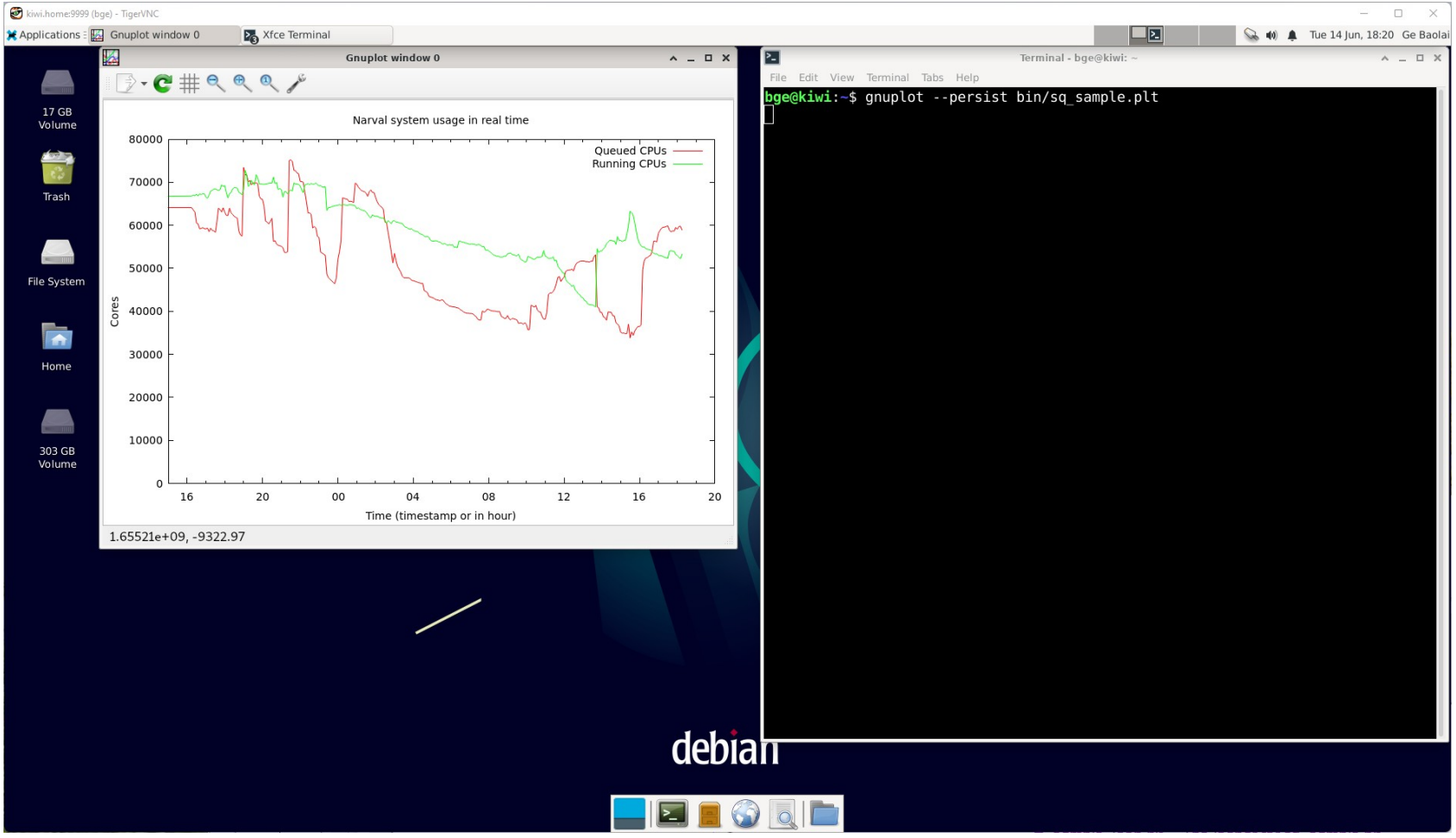
```
# "heat.plt" – Plot one frame at a time
set view 30,104,1,1
unset colorbox
set hidden3d

splot 'output.dat' using 1:2:3 with lines lt -1
reread
```

The Shell command

```
heatc | gnuplot –persist heat.plt
```

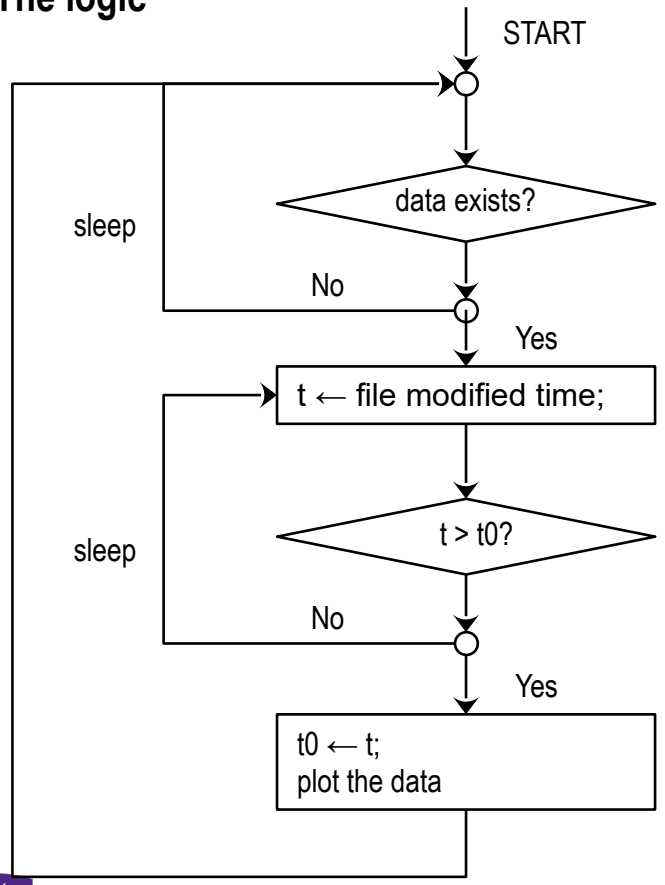
demo

Compute & display – asynchronously



Compute & display – asynchronously

The logic



Gnuplot scripts – *one graph window, persistent*

```
# "sample_loop.plt" - The looper for sq_sample.plt.
datafile = "data.csv"

t = system("expr `date +%s` - `stat -c %Y sq_data.csv`")
if (t == t0) {
   pause 30
   reread
}
t0 = t

plot datafile using 1:4 with lines lt rgb "red" title "Queued CPUs", "" using 1:5
with lines lt rgb "green" title 'Running CPUs'
reread
```

```
# “sample.plt” – The control script
t0 = 0
load 'sample_loop.plt'
```

The Shell command

```
gnuplot –persist sample.plt
```

demo

Compute & display – asynchronously

The workflow

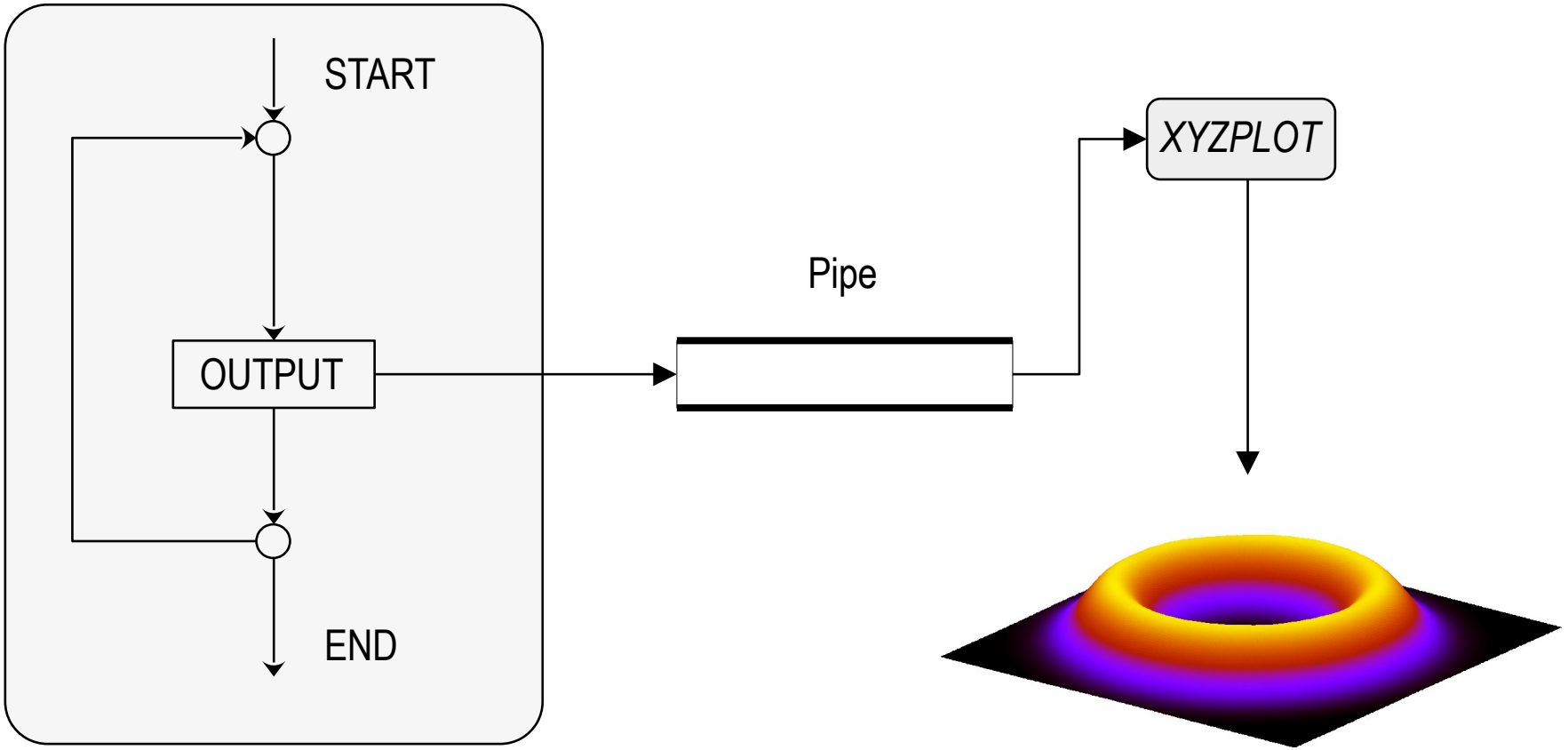
- The compute process produces a snapshot of output at a time periodically, either to the standard output or a file
- A separate, e.g. Gnuplot process, reads the output results as input and plots it as a “frame” in a sequence of display.
- **Pros:** No change in the compute program. Display control is done separately.
- **Cons:** The compute program and the graphical program run concurrently, the graphical program may read the output file that is just truncated to zero for writing by the program. So we could get a corrupted frames.

Meet the requirements?

- The graphical program can read the output from the compute program.
- The graphical program can generate multiple frames within the same figure window, make animation in real time possible.

Computing and visualizing synchronized via a pipe

Compute & display – synchronized via a pipe



Compute & display – synchronized via a pipe

The application code

```
// Open a pipe via system call popen(), creating a process of Gnuplot
FILE *fp=NULL;
fp = popen("gnuplot -persist","w");

// Read Gnuplot setup file from plot_cfg_file and write to the pipe
FILE *ff=NULL;
ff = fopen("heat_gnuplot.cfg","r");
… ...

// During the iteration, when generating the output, write the output to the pipe

   fprintf(fp, plot_command); // Write "splot - using 1:2:3 with pm3d"
   for (int j = 1; j <= nyg; j++)
   {
      for (int i = 1; i <= nxg; i++)
         fprintf(fp, "%f %f %f\n", xc[i], yc[j], unew[j][i]);
      fprintf(fp,"\n");
   }
   fprintf(fp,"e\n"); // Put an end mark to the stream
   fflush(fp);
```

Gnuplot script – *setup commands*

```
# "heat_gnuplot.cfg" – Gnuplot setup file
set term X11 size 1300, 720
unset parametric          # must set to parametric
set style data lines
set zrange[-1:1]
set view 40,120,1,1
set nokey              # no keys
unset border
unset xtics
unset ytics
unset ztics
set hidden3d
```

The Shell command

```
heatc
```

demo

Compute & display – synchronized via a pipe

SHARCNET™

Gnuplot

```
# “heat_gnuplot.cfg” – Gnuplot setup file
set term X11 size 1300, 720
                                    must set to parametric
                                    data lines
                                    e[-1:1]
                                    0,120,1,1
                                              # no keys
                                    der
                                    s
                                    s
                                    s
                                    n3d
```

Data stream in the pipe

splot - using 1:2:3 with pm3d

splot - using 1:2:3 with pm3d

splot - using 1:2:3 with pm3d

splot - using 1:2:3 with pm3d

Application

```
0.000000 0.000000 0.000000
0.015873 0.000000 0.000000
0.031746 0.000000 0.000000
0.047619 0.000000 0.000000
0.111111 0.000000 0.000000
0.126984 0.000000 0.000000
0.142857 0.000000 0.000000
… ...
```

demo

Compute & display – synchronized via a pipe

The workflow

- The compute process produces a snapshot of output at a time periodically, either to the pipe.
- A separate Gnuplot process reads the output results as input and plots it as a “frame” in the sequence of display.
- **Pros:** Compute and display are synchronized, graphical display is smooth. The data stream in the pipe is handled by the OS.
- **Cons:** The application program needs to open a pipe and close a pipe via system calls, which might not be explicitly supported, e.g. in Fortran. But that’s it.

Meet the requirements?

- The graphical program can read the output from the compute program from the pipe. The data once in the pipe no longer affected by the compute program. So no loss.
- The graphical program can generate multiple frames within the same figure window, make animation in real time possible.

Summary

Requirements	Direct, using third party library	Asynchronous, separate processes	Synchronous, via pipe
Display is platform independent	Maybe	No	Yes*
The display of output is smooth	Yes	Maybe	Yes
No missing “frames” or corrupted graphs	Yes	Not guaranteed	Yes
Difficulty to implement	Moderate to difficult	Easy	Easy
Does not slow down the compute part	Maybe	Maybe	Maybe

* For Windows, WSL2 supports pipe.

References

[1] SHARCNET training https://training.sharcnet.ca/

[2] Gnuplot on github http://www.gnuplot.info/