Beginner's guide to Basilisks

Vatsal Sanjay
vatsalsanjay@gmail.com

February 18, 2018

Abstract

This document is a review and guide to the installation and working of Basilisk (link), written by Stéphane Popinet and his group. These notes are for personal use only, and detailed guide could be found at <link>.

Regarding this file: Since basilisk is still in the early stages of development, the facts written in this document are subject to change. If you find any information here outdated or incomplete, please let me know. You can also clone this repository (https://github.com/VatsalSy/GuideBasilisk), make changes and push it.

1 Installing Basilisk

Installation is quite easy. There are two ways: Using darcs OR using tarball. I prefer the first one (darcs) because it allows to easily pull any update that occurs in Basilisk. The steps of this section are taken from http://basilisk.fr/src/INSTALL. They are self-explanatory I have included them here, just for the sake of completion.

1.1 Installing darcs (if unavailable)

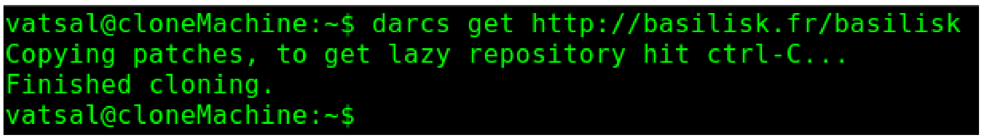
vatsal@cloneMachine:∼ $ sudo apt-get install darcs flex make

1.2 Some important side packages

vatsal@cloneMachine:∼ $ sudo apt-get install gnuplot imagemagick libav-tools smpeg-plaympeg graphviz valgrind gifsicle

1.3 Getting Basilisk source code

vatsal@cloneMachine:∼ $ darcs get http://basilisk.fr/basilisk



```
vatsal@cloneMachine:~$ darcs get http://basilisk.fr/basilisk
Copying patches, to get lazy repository hit ctrl-C...
Finished cloning.
vatsal@cloneMachine:~$
```

Note: In order to pull new changes made in Basilisk, use:
vatsal@cloneMachine:∼ $ cd basilisk
vatsal@cloneMachine:∼ $ darcs pull
Then recompile.

1.4 Compiling/recompiling Basilisk

vatsal@cloneMachine:∼ $ cd basilisk/src
vatsal@cloneMachine:∼ $ export BASILISK=$PWD
vatsal@cloneMachine:∼ $ export PATH=$PATH:$PWD
vatsal@cloneMachine:∼ $ ln -s config.gcc config
vatsal@cloneMachine:∼ $ make -k
vatsal@cloneMachine:∼ $ make

2 Basilisk-View

Details of bview: http://basilisk.fr/src/README#interactive-basilisk-view
Basilisk-view is similar to GfsView, with some minor differences:

1. Unlike GfsView, we cannot run bview on the fly while basilisk is running. We can, however, use it to render pictures and videos on the fly with bview. It can be used to look at the results after simulation is over, provided we have saved the intermediate files.
2. There is no program written analogous to gfs2oogl for bview. This means that we cannot interpolate the octree based data using available libraries. We could write our own code for it in future, something similar to gfs2oogl.
3. The coordinate systems for GfsView and Basilisk are different. GfsView output is rotated counter-clockwise compared to Gerris: this is due to the new N-ordering of quadtree cells (rather than Z-ordering). See these link1 and link2.
4. I found a working example for tecplot users (here), but I have not tested it yet.

GfsView can still be used with Basilisk, provided that the code does not use "mask"command. Mask command is used for creating solid objects and non-square (non-cubical) geometries. The precursor to using gfsview and writing ".gfs"files, "dump"is not compatible with "mask". Most likely, even bview cannot be used to look at intermediate files when "mask"is used, but I have not changed it. For other compatibility issues, visit: http://basilisk.fr/src/COMPATIBILITY.

2.1 Installing bview

Sources: (go in this order)

1. bview: http://basilisk.fr/src/bview
2. Screen rendering: http://basilisk.fr/src/gl/INSTALL
3. Python for bview-client: http://basilisk.fr/src/bview-client.py
4. Actual installation: http://basilisk.fr/src/bview-server.c

2.1.1 Dependencies

For cluster for off-screen rendering:

```
vatsal@cloneMachine:~ $ sudo apt-get install libglu1-mesa-dev libosmesa6-dev
vatsal@cloneMachine:~ $ cd $BASILISK/gl
vatsal@cloneMachine:~ $ make libglutils.a libfb_osmesa.a
```

Another method would be to use the following (recommended for laptops), the above dependencies are recommended for clusters.

```
vatsal@cloneMachine:~ $ sudo apt-get install libglu1-mesa-dev libglew-dev libgl1-mesa-dev
vatsal@cloneMachine:~ $ cd $BASILISK/gl
vatsal@cloneMachine:~ $ make libglutils.a libfb_glx.a
```

The following is required for both the above versions:

```
vatsal@cloneMachine:~ $ sudo apt-get install python-pil.imagetk
```

2.1.2 bview

(Optional) Add this line at the end of config.gcc, depending on the type of machine and dependency selected:
Osmesa (Cluster):

```
[frame=singhe]
OPENGLIBS = -lfb_osmesa -lGLU -lOSMesa
```

Glx (Laptop):

```
OPENGLIBS = -lfb_glx -lGLU -lGLEW -lGL -lX11
```

Then do this:

```
vatsal@cloneMachine:~ $ cd $BASILISK
vatsal@cloneMachine:~ $ make bview-servers
```

For a typical bview use and environment, follow the example at:http://basilisk.fr/src/bview

3 A typical code

Basilisk codes are modified C++ files.
A good place to start understanding Basilisk would be http://basilisk.fr/Tutorial. Here, I have added the case of liquid jet atomization. I have tried to compare and contrast this case with the famous jet atomization case simulated using gerris (which we are more familiar with).

3.1 Header File

Just like a standard C++ code, we need to include the libraries that we are going to use.

```
#include "navier-stokes/centered.h"
#include "two-phase.h"
#include "tension.h"
#include "tag.h"
#include "view.h"
```

1. *navier-stokes/centered.h* is the normal Gerris implemented Navier-Stokes equation solver for incompressible variable (two-phase) density flow.
2. *two-phase.h* is the simulations setup file for flows of two fluids separated by an interface (i.e. immiscible fluids). It is typically used in combination with a NavierStokes solver.
3. *tension.h* includes the surface tension force, same as Gerris by adding $\sigma\kappa\delta_i$ term in NS equation.
4. *tag.h* is used to count the number of droplets in the simulation. Unlike Gerris, in Basilisk, we get this special feature which when combined with a small sub-routine (given later) can count the number of droplets.
5. *view.h* is important for using Basilisk-View.

3.2 Defining custom variables

```
#define radius 1./12.
#define length 0.025
#define Re 5800
#define SIGMA 3e-5
int maxlevel = 7;
double uemax = 0.1;
```

Just like Gerris, here we can define some global variables which can be used later on in the file. For this specific example, radius is the radius of the liquid jet, length is the initialized length (similar to Gerris, we will initialize a small length of the jet and highly refine that part at t = 0), *Re* is the Reynold's number of the liquid jet, SIGMA is the σ (surface tension) at liquid-air interface, maxlevel is the maximum octree refinement level and uemax is the maximum allowable error in the velocity field.

3.3 Initial Conditions

```
scalar f0[];
//Initial conditions
event init (t = 0) {
  if (!restore (file = "dump")) {
    refine (x < 1.2*length && sq(y) + sq(z) < 2.*sq(radius) && level < maxlevel);
    fraction (f0, sq(radius) - sq(y) - sq(z));
    f0.refine = f0.prolongation = fraction_refine;
    restriction ({f0});
    foreach() {
      f[] = f0[]*(x < length);
      u.x[] = f[];
    }
    boundary ({f,u.x});
  }
}
```

A scalar field $f0$ is defined. It is analogous to the $T0$ variable we usually define in Gerris. An event instance is declared for declaring initial conditions. The commands are executed if and only if the file named “dump”is not present. The grid is refined in a cylindrical region, 1.2 times the size of initialized jet and $\sqrt{2}$ times its radius. Further, the variable field $f0$ is set at 1 inside the cylinder ($y^2 + z^2 < r^2$) and zero otherwise. Moreover, *f0.refine = f0.prolongation = fraction_refine;* refines the volume fraction field (details). Further, *restriction(f0);* is used for boundary conditions on levels. The *foreach* command is a modified *for* loop to initialize the initial conditions in all the cells. At last, *boundary (f,u.x);* defines the boundary conditions at t = 0 (details). The stencils located near the boundary extends beyond it and the last line initializes these ghost values. By default, the boundary condition is symmetry and if the domain values are modified, it is necessary to modify the ghost values as well and therefore the last line is needed.

3.4 Boundary Condition

```
/* We then set an oscillating inflow velocity on the
left-hand-side and free outflow on the right-hand-side. */
u.n[left]  = dirichlet(f0[]*(1. + 0.05*sin (10.*2.*pi*t)));
u.t[left]  = dirichlet(0);
p[left]    = neumann(0);
f[left]    = f0[];
u.n[right] = neumann(0);
p[right]   = dirichlet(0);
```

Setting boundary conditions is intuitive and the above code snip is self-explanatory. Unlike Gerris, we do not have to create the web of boxes and manually see to the inter-connectivities. In order to get a cuboidal (rectangular) domain or create complex geometries, “mask”can be used. However, “dump”command used to write output data is not compatible with “mask”yet.

3.5 Adaptive Mesh Refinement

```
event adapt (i++) {
  adapt_wavelet ({f,u}, (double[]){0.01,uemax,uemax,uemax}, maxlevel);
}
```

Basilisk uses the “wavelet”method for AMR based on the maximum allowable error. In the syntax, {*f*,*u*} defines the variables based on which refinement is to be done (please note, it is a 4 element vector). In the second term, {0.01,uemax,uemax,uemax} enlists the maximum error allowed.

3.6 The end time of simulation

Unlike Gerris, there is no option to set end-time in Basilisk (that I could find). It is set with the maximum time event available in the script. Therefore, we can make one event to be executed at the end time of simulation and do not mention time anywhere else in the code. Please note that the “event”is

```
event end (t = 2.5) {
  printf ("i = %d t = %g\n", i, t);
}
```

3.7 The main run snippet

```
int main ()
{
  /**
  The initial domain is discretised with $64^3$ grid points. We set
  the origin and domain size. */
  init_grid (64);
  origin (0, -1.5, -1.5);
  L0 = 3.;
  /*
  We set the density and viscosity of each phase as well as the
  surface tension coefficient and start the simulation. */
  rho1 = 1., rho2 = 1./27.84;
  mu1 = 2.*radius/Re*rho1, mu2 = 2.*radius/Re*rho2;
  f.sigma = SIGMA;
  run();
}
```

This is the main function which executes the simulation. The command *init_grid (n)* descritizes the domain length in n number of cells.

Origin: Setting origin is different than that in Gerris. Origin can be set using *origin* $(x_0,\ y_0,\ z_0)$. In this, $(x_0,\ y_0,\ z_0)$ is the co-ordinate location of the left-bottom-back corner with respect to the intended origin. For example, if you want the origin to be at center of the left wall, the command will read (0, -1.5, -1.5).
L0 is used to set the length of the domain. Please note that the domain is always cubical in Basilisk. So if you want to make a cuboid then it is necessary to use "mask". The other variables, *mi* and *rhoi* ($\forall\ i \in \{1, 2\}$) define the viscosity and density respectively and are local variables to the file "twophase.h". At last, *f.sigma* is the surface tension for the VOF tracer *f*.

3.8 Writing log files and on terminal

```
event logWriting (i++) {
  static FILE * fp;
  if (i == 0) {
    fprintf (ferr, "i dt t\n");
    fp = fopen ("log", "w");
    fprintf (fp, "i dt t\n");
    fprintf (fp, "%d %g %g\n", i, dt, t);
    fclose(fp);
  } else {
    fp = fopen ("log", "a");
    fprintf (fp, "%d %g %g\n", i, dt, t);
    fclose(fp);
  }
  fprintf (ferr, "%d %g %g\n", i, dt, t);
}
```

This part is used to output the simulation temporal information on the terminal as well as store them in the file named "log".

3.9 Generating movies

```
/**
We generate an animation using Basilisk View. */
event movie (t += 1e-2)
{
#if dimension == 2
  scalar omega[];
  vorticity (u, omega);
  view (tx = -0.5);
  clear();
  draw_vof ("f");
  squares ("omega", linear = true, spread = 10);
  box ();
#else // 3D
  scalar pid[];
  foreach()
    pid[] = fmod(pid()*(npe() + 37), npe());
  boundary ({pid}); // not used for the moment
  view (camera = "iso",
fov = 14.5, tx = -0.418, ty = 0.288,
width = 1600, height = 1200);
```

```
  clear();
  draw_vof ("f");
  box ();
#endif // 3D
  static FILE * fp = popen ("ppm2mp4 > movie.mp4", "w");
  save (fp = fp);
}
```

This is just a typical code to generate movie of the simulation. For details, look at the guide to bview.

3.10 Data dumping

```
event snapshot (t = 0; t += 0.1)
{
  dump (file = "dump");
  char nameGfs[80];
  sprintf (nameGfs, "intermediateGfs/snapshot-%2.1f.gfs", t);
  output_gfs (file = nameGfs, t = t);
  char nameBview[80];
  sprintf (nameBview, "intermediateBview/snapshot-%2.1f", t);
  dump (file = nameBview);
}
```

These are used to write data files. To write ".gfs"files, *output_gfs (file = nameGfs, t = t);* is used. For writing data files read and processed by bview, "dump (file = nameBview);'can be used.

3.11 Visualization on the fly

Currently, bview is not equipped for visualization on the fly. But, the following code can be used to run gfsview on the go.

```
event gfsview (i++) {
  static FILE * fp = popen ("gfsview3D", "w");
  output_gfs (fp);
}
```

Please note that in presence of "mask", the "dump"and "output_gfs"commands will write corrupted intermediate files.

3.12 Running scripts

3.12.1 At t = 0

To start the simulation, use the script file, named "run0.sh". It deletes the log and dump files if they exist and creates the empty folders.

```
if [ -f "log" ];
then
rm log
fi

if [ -f "dump" ];
then
rm dump
fi

if [ -d "intermediateGfs" ]
then
rm -r intermediateGfs
fi

if [ -d "intermediateBview" ]
```

```
then
rm -r intermediateBview
fi

mkdir intermediateGfs
mkdir intermediateBview

qcc -O2 -Wall -grid=octree $file.c -o $file -lm \
    -L$BASILISK/gl -lglutils -lfb_glx -lGLU -lGLEW -lGL -lX11 -lm

#qcc -O2 -Wall $file.c -o $file -lm \
#    -L$BASILISK/gl -lglutils -lfb_osmesa -lGLU -lOSMesa -lm
```

Moreover, in order to make changes in between the simulation and re-start it, the script file named “runRe.sh”can be used. This simply compiles the code.

```
if [ ! -d "intermediateGfs" ]
then
mkdir intermediateGfs
fi

if [ ! -d "intermediateBview" ]
then
mkdir intermediateBview
fi

qcc -O2 -Wall -grid=octree $file.c -o $file -lm \
    -L$BASILISK/gl -lglutils -lfb_glx -lGLU -lGLEW -lGL -lX11 -lm

#qcc -O2 -Wall $file.c -o $file -lm \
#    -L$BASILISK/gl -lglutils -lfb_osmesa -lGLU -lOSMesa -lm
```