Fractal Gravity Reference v. 1.1.1

Jens Verner Villumsen May 5, 2018

This paper has the necessary routines and parameters to call FractalGravity with default parameters. This is for use with both periodic and isolated boundary conditions.

For now, only isolated boundary conditions are implemented.

It is organized in what I consider a logical order. Later versions will include description of parameters you can change.

FractalGravity is memory intensive due to the author's inexperience when he started this project. The memory requirements will be reduced in later versions.

All routines and variables are in namespace FractalSpace.

All communication with FractalGravity is done through a single pointer to an object of the Fractal_Memory class.

If you are not familiar with FractalGravity, use the default parameters and ignore the section about Optional Simulation Parameters.

Program Flow

The first two calls are setup calls to

FractalGravityFirstTime

fractal_memory_setup

They are called only once before calling the solver.

Before each call to the solver you call

setNumberParticles

fractal_create

add_particles

FractalCube

balance_by_particles

The call to the solver is

DoFractalGravity

To get your data back you call

get_field

To end the step you must call

fractal_delete

After the last call to the solver you call

fractal_memory_content_delete

fractal_memory_delete

The only thing you need to remember is the pointer to the Fractal_Memory object.

Description of Routines

FractalGravityFirstTime

```
FractalSpace::Fractal_Memory* FractalSpace::FractalGravityFirstTime(
                                                    bool Periodic,
                                                     MPI_Comm& TalkToMe,
                                                     int GridLength,
                                                     int FractalNodes0,
                                                     int FractalNodes1,
                                                     int FractalNodes2,
                                                     string BaseDirectory,
```

string RunIdentifier);

This call creates the Fractal_Memory object. It must be called before any other FractalGravity calls. This routine is called only once.

Return value "Fractal_Memory* PFM" is a pointer to the Fractal_Memory class object.

It is the only pointer you need to remember.

Bool Periodic	periodic=true/isolated=false boundary conditions
MPI_Comm& TalkToMe,	Communicator for the nodes you give FractalGravity.
int GridLength,	Length of the cubical Fourier grid. Preferably a power of two. Even better, a power of 4. Must be a product of powers of small primes GridLength=2^a*3^b*5^c. You should use the largest possible value of GRIDLENGTH subject to memory constraints because FFTW is faster than HYPRE. A larger value also helps the load balancing.
int FractalNodes0,	x-length of the pseudo Cartesian grid of nodes.
int FractalNodes1,	y-length of the pseudo Cartesian grid of nodes.
int FractalNodes2,	z-length of the pseudo Cartesian grid of nodes

FractalNodes0 x FractalNodes1 x FractalNodes2 must equal the number of nodes in TalkToMe. FractalNodes0, FractalNodes1, FractalNodes2 can be any integers greater than one. For efficiency, they should be close. 8x4x4 is better than 8x8x2. FractalNodes0 >= FractalNodes1 >= FractalNodes2 improves efficiency in many situations.

string BaseDirectory,	Absolute path of base for output directories. Example /p/lscratche/jensv/galaxy/
string RunIdentifier	Label for this run Example NerdsRule

If N is the number of nodes, then node "n" will write to directory "BaseDirectory/N/RunIdentifier_%_n" where % is a capital letter.

Directory "Basedirectory”N”/" must exist before calling this routine.

This will generate directory /p/lscratche/jensv/galaxy/256/NerdsRule_C_17 for node# 17 if we have already run two simulations with 256 nodes. The next simulation will write on NerdsRule_D_17.

fractal_memory_setup

void FractalSpace::Fractal_Memory_setup(FractalSpace::Fractal_Memory* PFM)

"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime.

This routine is called only once after FractalGravityFirstTime but before any other FractalGravity calls. It sets up everything the simulation needs. *PFM has a large number of parameters that are member variables of *PFM. For now you can use the default variables. If you want to change any member variable this must be done before calling this routine.

setNumberParticles

Before you create fractal you must tell *PFM how many particles you are going to give it. Sorry, legacy issues.

FractalSpace::PFM->setNumberParticles(int NumberParticles)

	"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime.
int NumberParticles	Number of particles in this particular step.

fractal_create

void FractalSpace::fractal_create(FractalSpace::Fractal_Memory* PFM);

"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime. It creates an object of the Fractal class that is used internally to hold particle information. It must be called before each call to DoFractalGravity.

add_particles

void FractalSpace::add_particles(FractalSpace::Fractal_Memory* PFM,int first,int total,

vector <double>& posx,vector <double>& posy,

vector <double>& posz,vector <double>& masses);

Add the particle positions and masses of particles in your units. You must add particles from "0" to "PFM->NumberParticles-1". You can do this in multiple stages.

PFM	Pointer to the Fractal_Memory class object you created in FractalGravityFirstTime.
first	First particle you add to *PFM.
total	Number of particles you add to PFM
posx	x-coordinates in your units
posy	y-coordinates in your units
posz	z-coordinates in your units
masses	masses of particles in your units

FractalCube

FractalSpace::FractalCube(FractalSpace::Fractal_Memory* PFM,double SHRINK,
vector <double>& xmin, vector <double>& xmax,
vector <double>& xmini, vector <double>& xmaxy)

PFM	Pointer to the Fractal_Memory class object you created in FractalGravityFirstTime.
SHRINK	If SHRINK <= 0.0, Fractal uses the supplied cube. If SHRINK >= 1.0, Fractal uses the smallest cube holding all particles. Else, the routine will find a compromise between the two cubes. For now use either 0.0 or 1.0 for SHRINK. Long story.
xmin	Lower left corner(xa,ya,za) of supplied computational cube
xmax	Upper right corner(xb,yb,zb) of supplied computational cube. yb,zb are not used
xmini	Lower left corner(xia,yia,zia) of calculated computational cube
xmaxy	Upper right corner(xyb,yyb,zyb) of calculated computational cube. yyb,zyb are not used

Routine find the smallest cube that holds all particles. The cube can be anywhere in space. If the calculated size of the cube is larger than the supplied cube, the supplied cube will be used. Particles outside the cube are ignored by FractalGravity. It helps to make the cube as small as possible even if a few of the particles are left outside. It makes load balancing easier and Hypre faster.

This routine could use some serious work.

balance_by_particles

void FractalSpace::balance_by_particles(FractalSpace::Fractal_Memory* PFM,bool withparts)

Each node is assigned a box in space. The set of nodes fills the computational cube. The routine tries to equalize the volumes of nodes or the number of particles on the nodes

PFM	Pointer to the Fractal_Memory class object you created in FractalGravityFirstTime.
withparts	If false, equal volume on all nodes.
	If true, equal number of particles. Recommended.

This routine could use some serious work.

DoFractalGravity

void FractalSpace::DoFractalGravity(FractalSpace::Fractal_Memory* PFM);

"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime. Do the actual calculation of the gravitational potentials and accelerations.

get_field

void FractalSpace::get_field(FractalSpace::Fractal_Memory* PFM,int first,int total,double G_Cavendish,

vector <double>& xmini,vector <double>& xmaxy,

vector <double>& pot,vector <double>& accx,

vector <double>& accy,vector <double>& accz);

Return the gravitational potentials and accelerations in your units. You can make this call multiple times.

PFM	Pointer to the Fractal_Memory class object you created in FractalGravityFirstTime.
first	First particle you want data for.
total	Number of particles you want data for.

G_Cavendish	Your gravitational constant.
xmini	Lower left corner(xia,yia,zia) of computational cube
xmaxy	Upper right corner(xyb,yyb,zyb) of computational cube. yyb,zyb are not used.
pot	Gravitational potential.
accx	x-acceleration
accy	y-acceleration
accz	z-acceleration

fractal_delete

void FractalSpace::fractal_delete(FractalSpace::Fractal_Memory* PFM);

"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime. Deletes the fractal object and all particle data.

fractal_memory_content_delete

void FractalSpace::fractal_memory_content_delete(FractalSpace::Fractal_Memory* PFM);

"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime. Deletes the contents of the Fractal_Memory object.

fractal_memory_delete

void FractalSpace::fractal_memory_delete(FractalSpace::Fractal_Memory* PFM);

"PFM" is a pointer to the Fractal_Memory class object you created in FractalGravityFirstTime. Deletes the Fractal_Memory object and sets the PFM pointer to 0.

Optional Simulation Parameters

FractalGravity has many parameters, most of which you should not worry about. You can change parameters between calls to DoFractalGravity. All parameter changes must be done before the fractal_create call.

FractalSpace::PFM->setLevelMax(int LevelMax); Default: int LevelMax=8;

The number of levels in the Tree of Points. If you want to limit the resolution, set LevelMax to a lower number. It is inexpensive in time and memory to go beyond level=2.

FractalSpace::PFM->setMinimumNumber(int MinimumNumber); Default: int MinimumNumber=8;

A Point defines a volume on a node. If this volume has at least MinimumNumber of particles, the volume is split into 8 subvolumes. Increasing MinimumNumber will lower the volume calculated at high resolution. Decreasing MinimumNumber will increase the volume calculated at high resolution and may cause a runaway.

void FractalSpace::PFM->setBalance(int Balance); Default: int Balance=1;

If Balance==0, all nodes have the same volume, approximately.

If Balance==1, all nodes have the same number of particles, approximately.

void FractalSpace::PFM->setPadding(int Padding); Default: int Padding=-1;

If Padding==0, no padding of HighPoints. Speeds up the code and lowers memory requirements. Increased noise and possibly rapid change in spatial resolution.

If Padding=-1, HighPoints are padded so that resolution cannot change by more than a factor 2 across a boundary. Economical.

If Padding==1, HighPoints are fully padded. Resolution cannot change by more than a factor 2 across a boundary. Low noise but expensive in time and memory.

FractalSpace::PFM->setHypreIterations(int MaxHypreIterations); Default: int MaxHypreIterations=20;

Maximum number of iterations for Hypre PCG solver. The solver will output its best estimate of the solution.

FractalSpace::PFM->setHypreTolerance(double HypreTolerance); Default: HypreTolerance=1.0e-7;

Set the relative convergence tolerance.

Header Files:

libs.hh	The C and C++ libraries I use. Should be cleaned up.
classes.hh	My classes
headers.hh	My headers. Should be cleaned up.
fractal_interface_public.hh	Interface headers