Geophysical Fluids Modeling Framework Handbook

GAME Development Team

Contents

Chapter 1

Overview

The Geophysical Fluids Modeling Framework (GAME) is a non-hydrostatic hexagonal C-grid dynamical core with the possibility to advect a variable number of constituents. The term *dynamical core* typically refers to the simulation of a dry atmosphere, including horizontal momentum diffusion. Everything else is then referred to as *physics*. Diffusive terms, including turbulence parameterizations, are sometimes understood to be part of the dynamical core and sometimes seen as part of the model's physics. The dry air is in this understanding a "carrier medium", whereas constituents, including water in different phases, are usually only passively advected. This thinking always leads to deep physical inconsistencies during later stages of model development, whose impact on forecast and climate simulation accuracy remains unknown.

Therefore, a new, capaple framework for simulating geophysical fluids is necessary and, due to the advent of even more powerful computers, also realistic. GAME can be seen as a dynamical core, but in a modernized sense. Its aim is to simulate the dynamics of geophysical fluid flow accurately, and at the same time make it possible to couple the model to different constituents *self-consistently*, which means without violating fundamental physical constraints. For radiation, it is coupled to the RTE+RRTMGP (Radiative Transfer for Energetics + Rapid and Accurate Radiative Transfer Model for Geophysical Circulation Model Applica [**doi:10.1029/2019MS001621**], [**rte-rrtmgp-github**] scheme, which follows a similar approach to radiation simulation as GAME follows to fluid simulation.

This is only the handbook (manual) of the Geophysical Fluids Modeling Framework (GAME), it explains how to configure, compile and run (use) the model. For a scientific derivation of the model see [**kompendium**] and the literature cited therein. The source code of the project is maintained on Github (https://github.com/OpenNWP/GAME).

The GAME project incorporates two different executables:

- grid_generator, a program for creating model grids
- game, the model executable itself

Chapter 2

Code structure

The code of the model resides in the directory `src`.

2.1 Spatial operators

- Coriolis: [**thuburn_f_discrete_plane**] and [**ringler_trsk**] modified by [**doi:10.1002/qj.3294**]
- kinetic energy: [**doi:10.1002/qj.1960**]

2.2 Time stepping

A fully Eulerian time stepping is employed. The basic building structure is a two-time-level predictor-corrector scheme. In the vertical, at every substep, an implicit column solver is used, which makes it possible to violate the CFL criterion of vertically propagating sound and fast gravity waves. This has the cost of decreasing the accuracy of these modes, which is however a bearable trade-off, since these waves are of low meteorological relevance. Furthermore, a forward-backward scheme is used, where the divergence term is backward.

Chapter 3

Installation

3.1 Dependencies

The following dependencies must be installed before being able to successfully build the model:

- `sudo apt-get install gfortran make cmake wget python3-pip libnetcdff-dev`
- Clone the DCMIP2016 repository: `git clone https://github.com/ClimateGlobalChange/DCMIP2016`
- Clone our fork of the RTE+RRTMGP repository: `git clone https://github.com/OpenNWP/rte-rrtmgp`
- `pip3 install global-land-mask`
- The Python visualization library scitools-iris (installation manual: `https://scitools-iris.readthedocs.io/en/stable/installing.html#installing-from-source-without-conda-on-debian-based-linux-distros-developers`, only for the plotting routines)
- FFMPEG (Ubuntu: `sudo apt-get install ffmpeg`, only for the plotting routines)

3.2 Building

`CMake` is used for building `GAME`. Execute `./compile.sh` to build the model.

Chapter 4

Grid generation

4.1 Fundamental grid quantities

The following quantities are sufficient to uniquely define the grid:

- the horizontal coordinates of the generating points
- the numbering of the generating points
- the numbering and orientation of the horizontal vectors
- the numbering of the dual cell mid points
- The orientation of the dual horizontal vectors. Since their horizontal positions coincide with the horizontal positions of the primal vectors, both sets of vectors can be numbered in the same way.

All other quantities, be it floating point numbers like grid box volumes or areas, or integer quantities like neighborhood relationships, can be implicitly derived. Consequently, once must firstly focus on the fundamental grid properties.

4.1.1 Creating a spherical geodesic grid

By *horizontal grid* we mean the grid on one single layer. Without loss of generality this layer can be assumed to coincide with the surface of the unity sphere.

4.1.2 Numbering and orienting the vector points

4.1.3 Numbering and orienting the dual vector points

Until now, we have only operated with the six following arrays:

- `latitude_scalar`, `longitude_scalar`
- `from_index`, `to_index`
- `from_index_dual`, `to_index_dual`

4.2 Derived quantities

4.2.1 Horizontal coordinates of dual scalar points

The dual cells of a hexagonal grid are the triangular grid cells. Since we aim at an orthogonal grid, the dual scalar points must be the Voronoi centers of the triangular cells. Label the vertices of one triangle with $\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2$, then its Voronoi center $\mathbf{r}_v$ is located at the position

$$\mathbf{r}_v \coloneqq \frac{(\mathbf{r}_1 - \mathbf{r}_0) \times (\mathbf{r}_2 - \mathbf{r}_0)}{|(\mathbf{r}_1 - \mathbf{r}_0) \times (\mathbf{r}_2 - \mathbf{r}_0)|}. \tag{4.1}$$

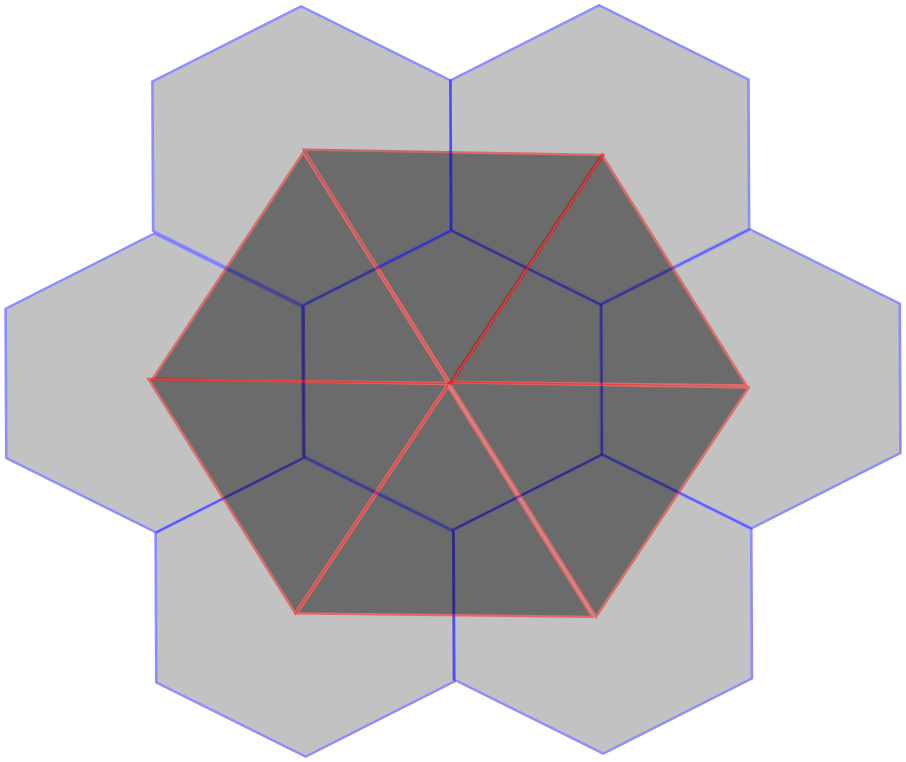

Figure 4.1: A subset of a regular horizontal hexagonal grid. The hexagonal grid (blue lines) and the triangular grid (red lines) form a pair of a primal-dual grid. In GAME, the hexagonal grid is the primal grid, while the triangulars form the dual one. In a triangular grid model it is the other way around. During the grid generation procedure we refer to the triangle edge points (hexagon centers) as the generating points or generators, for short.

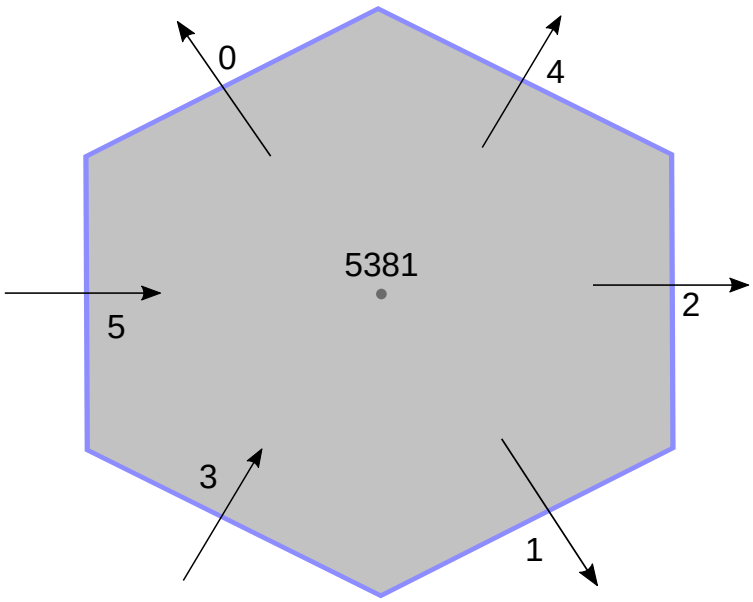

Figure 4.2: A sample hexagon with a horizontal scalar index of 5381. The directions of the arrows indicate the directions of unit vectors at cell edges. The drawn orientations would lead to `adjacent_vector_signs_h`$[6\cdot5381+0]=1$, `adjacent_vector_signs_h`$[6\cdot5381+1]=1$, `adjacent_vector_signs_h`$[6\cdot5381+2]=1$, `adjacent_vector_signs_h`$[6\cdot5381+3]=-1$, `adjacent_vector_signs_h`$[6\cdot5381+4]=1$, `adjacent_vector_signs_h`$[6\cdot5381+5]=-1$.

4.2.2 Finding the neighboring vector points of a primal cell and their orientation

4.2.3 Finding the neighboring vector points of a dual cell and their orientation

4.3 Grid optimization

Hexagonal spherical grids need to be optimized for numerical modeling. Therefore, the Lloyd algorithm is used, which yields a *spherical centroidal Voronoi tesselation (SCVT)* after convergence [**Du2003**]. [**PEIXOTO201361**] gives an overview of optimization alternatives and it seems to be that the SCVT is the most suitable for modeling. The procedure employed for executing the Lloyd algorithm is the one described in [**10.1175/MWR2991.1**].

4.4 Vertical grid structure

So far, only a horizontal grid has been examined. The grid generator, however, shall produce full three-dimensional grids. In order to simplify matters, the following conventions are made:

- Since the vertically oriented primal vector points have the same horizontal coordinates as the primal scalar points, their horizontal numbering is also the same.
- Since the vertically oriented dual vector points have the same horizontal coordinates as the dual scalar points, their horizontal numbering is also the same.

4.5 Scalability

The computation time of the most expensive for loops scale with N^2, where N is the number of horizontal grid points. This means that doubling the horizontal resolution (four times as much horizontal grid points) leads to a 16 times longer computation time of the grid generator. This is similar to the model itself, where a doubling of the horizontal and vertical resolution and a halfening of the time step leads to 16 times longer integration times. Therefore, the largely implicit formulation of the grid generator poses no problem to its performance at higher resoultions.

4.5.1 Permutations of the grid points

4.6 Horizontal grid properties

The horizontal grid structure is determined by the following properties:

- the resolution, specified via the parameter `RES_ID`
- the optimization

4.7 Vertical grid properties

The vertical grid structure is determined by the following properties:

- the height of the top of the atmosphere, specified via the parameter `TOA`
- the number of layers, specified via the parameter `NO_OF_LAYERS` N_L
- the number of layers following the orography, specified via the parameter `NO_OF_ORO_LAYERS` N_O
- the stretching parameter β, which can be set in the run script
- the orography, specified via the parameter `oro_id`

The generation of the vertical position of the grid points works in three steps:

1. First of all, vertical positions of preliminary levels with index $0 \le j \le N_L$ are determined by

$$z_j = T\sigma_{z,j} + B_j z_S, \tag{4.2}$$

where T is the top of the atmosphere, $\sigma_{z,j}$ is defined by

$$\sigma_{z,j} := \left(1 - \frac{j}{N_L}\right)^{\alpha}, \tag{4.3}$$

where $\alpha \ge 1$ is the so-called *stretching parameter*, z_s is the surface height and B_j is defined by

$$B_j := \frac{j - (N_L - N_O)}{N_O}. \tag{4.4}$$

name	domain	meaning
oro_id	all value for which an orography is defined	orography ID
n_iterations	integer ≥ 1	number of iterations (ignored if optimize = 0), 2000 seems to be a safe value
use_scalar_h_coords_file	0, 1	switch to determine wether horizontal coordinates of triangle vertices (generators of the grid) shall be used from a previously generated file
scalar_h_coords_file	string	input file for dual triangle vertices (only relevant if use_scalar_h_coords_file = 1)
stretching_parameter	≥ 1, real	defines the vertical stretching of the grid, one means no stretching
toa	> 0, real	height of the top of the atmosphere
type_of_vertical_grid	0, 1	specifies the terrain handling; 0: terrain-following coordinates, 1: block-like orography
orography_layers	≥ 1, natural	number of layers following orography (only relevant if type_of_vertical_grid == 0)
radius_rescale	> 0, real	rescale factor for the radius of the grid, radius r will be calculated according to $r =$ radius_rescale $\cdot a$, where a is the Earth radius; angular velocity ω will be replaced according to $\omega \rightarrow \frac{\omega}{\text{radius\_rescale}}$
no_of_avg_points	≥ 1	number of points used for smoothing the orography

Table 4.1: Grid generator run script explanation.

2. Then, the scalar points are positioned in the middle between the adjacent preliminary levels.
3. Then, the vertical vector points are regenerated by placing them in the middle between the two adjacent layers.
4. Finally, the vertical positions of the other points are diagnozed through interpolation.

4.8 How to generate a grid

The grid generator needs to be recompiled for every specific horizontal resolution and number of layers. In order to do that, change the respective constants in the file src/game_types.h and execute the bash script grid_generator/compile.sh. Then run the grid generator using the bash script grid_generator/run_script.sh with the desired oro_id. Table 4.1 explains all the parameters to be set in grid_generator/run_script.sh.

4.9 Physical surface properties

The properties of the surface of the Earth influence the evolution of the atmospheric fields. Therefore, physical surface properties need to be obtained from external sources and interpolated to the model grid. The following fields are required:

- The land-sea mask. It is needed because the model needs to know in which grid box water is present in order to calculate the evaporation and sublimation rates at the surface. data source: Python package global-land-mask
- The orography. It is needed because the model needs to calculate the interactions of the flow with the mountains. source: https://www.ngdc.noaa.gov/mgg/global/relief/ETOPO1/data/ice_surface/grid_registered/netcdf/ETOPO1_Ice_g_gmt4.grd.gz
- The density of the soil.
- The specific heat capacity of the soil.

oro_id	**Description**
0	no orography
1	real orography

Table 4.2: Definition of orography IDs.

- The temperature diffusivity of the soil.
- For NWP runs without coupling to an ocean model, the SST needs to be prescribed in order to calculate sensible and latent heating rates at the ocean surface. source: `https://ftp.ncep.noaa.gov/data/nccf/com/gfs/prod` (The SST can also be set analytically for idealized simulations.)

Tab. 4.2 shows the definition of the orography IDs. Before creating a grid file with real orography (orography ID 1), you have to create a grid file of orography ID 0.

Chapter 5

Ideal test states

TEST_ID	**Description**
0	standard atmosphere
1	dry Ullrich test
2	moist Ullrich test

Table 5.1: Definition of test IDs.

Chapter 6

Running the model

The configuration of the model must be set in two different files:

- `src/game_types.h`: modify `RES_ID`, `NO_OF_LAYERS`, `NO_OF_GASEOUS_CONSTITUENTS` and `NO_OF_CONDENSED_CONSTITUENTS`. `RES_ID` and `NO_OF_LAYERS` must conform with the grid file. It must be done before the compilation.
- The run script: one of the files contained in the directory `run_scripts`. The comments in these files explain the meaning of the variables. This can be done at run time.

6.1 Dynamics configuration

6.2 Physics configuration

6.3 Coupling to the radiation field

`GAME` employs the so-called `RTE+RRTMGP (Radiative Transfer for Energetics + Rapid and Accurate Radiative Transfer Model for Geophysical Circulation Model Applications-Parallel)` [**doi:10.1029/2019MS001621**], [**rte-rrtmgp-github**] schem

6.4 Configuring output

6.4.1 Output on pressure levels

In meteorology, atmospheric fields are often visualized on pressure levels. To simply this, `GAME` can interpolate data to pressure levels. This can be turned on and off with the variable `pressure_level_output_switch` in the run scripts.

If you want to change the pressure levels to which the output will be interpolated, modify the function `get_pressure_levels` as well as the constant `NO_OF_PRESSURE_LEVELS` in the file `src/io/write_output.c`. This must be done before compiling the model.