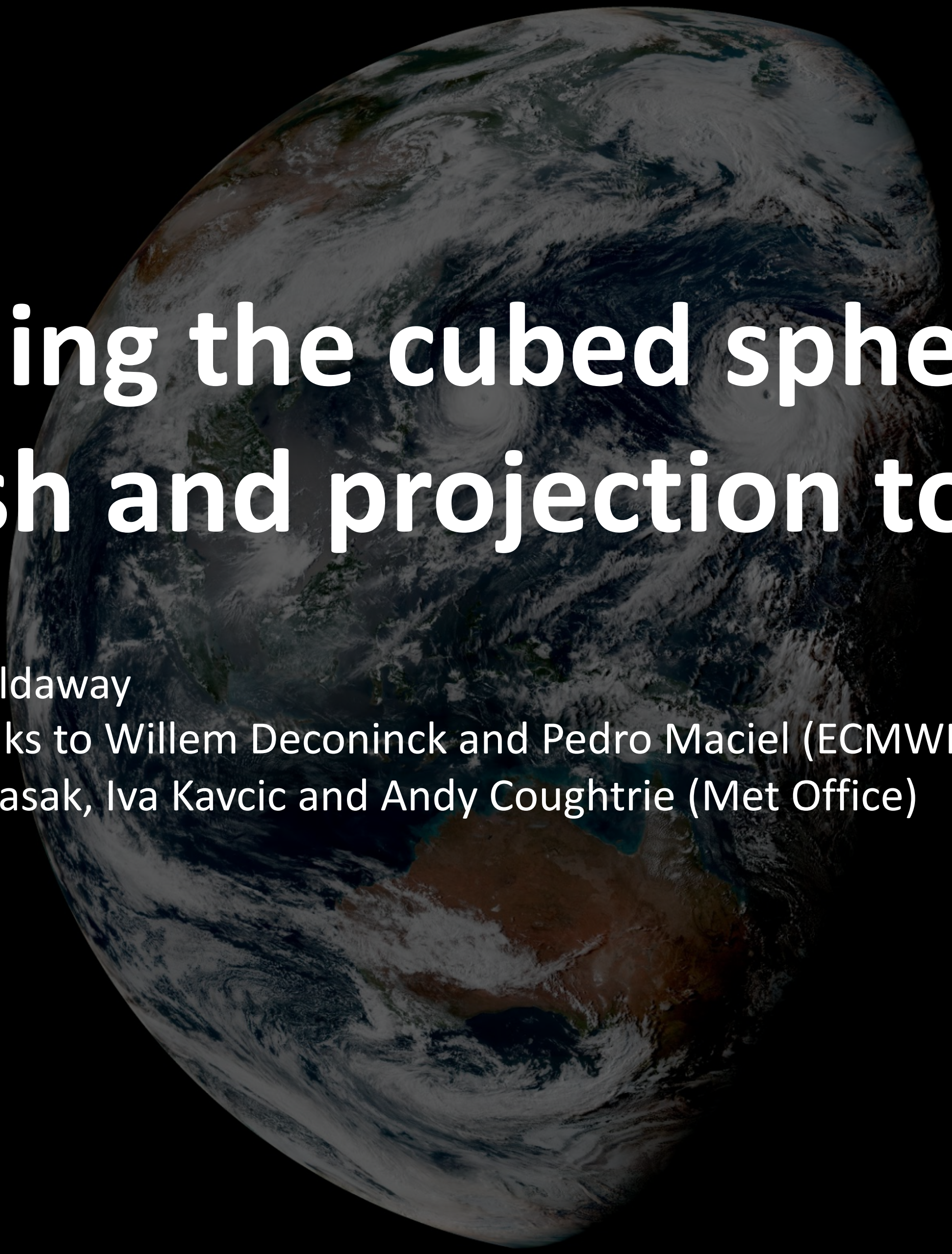


Adding the cubed sphere grid, mesh and projection to Atlas

Daniel Holdaway
With thanks to Willem Deconinck and Pedro Maciel (ECMWF) and
Marek Wlasak, Iva Kavcic and Andy Coughtrie (Met Office)

Why Atlas for the Model interface

- Using Atlas for the interface will allow for more generic Geometry, State, Increment and GetValues classes.

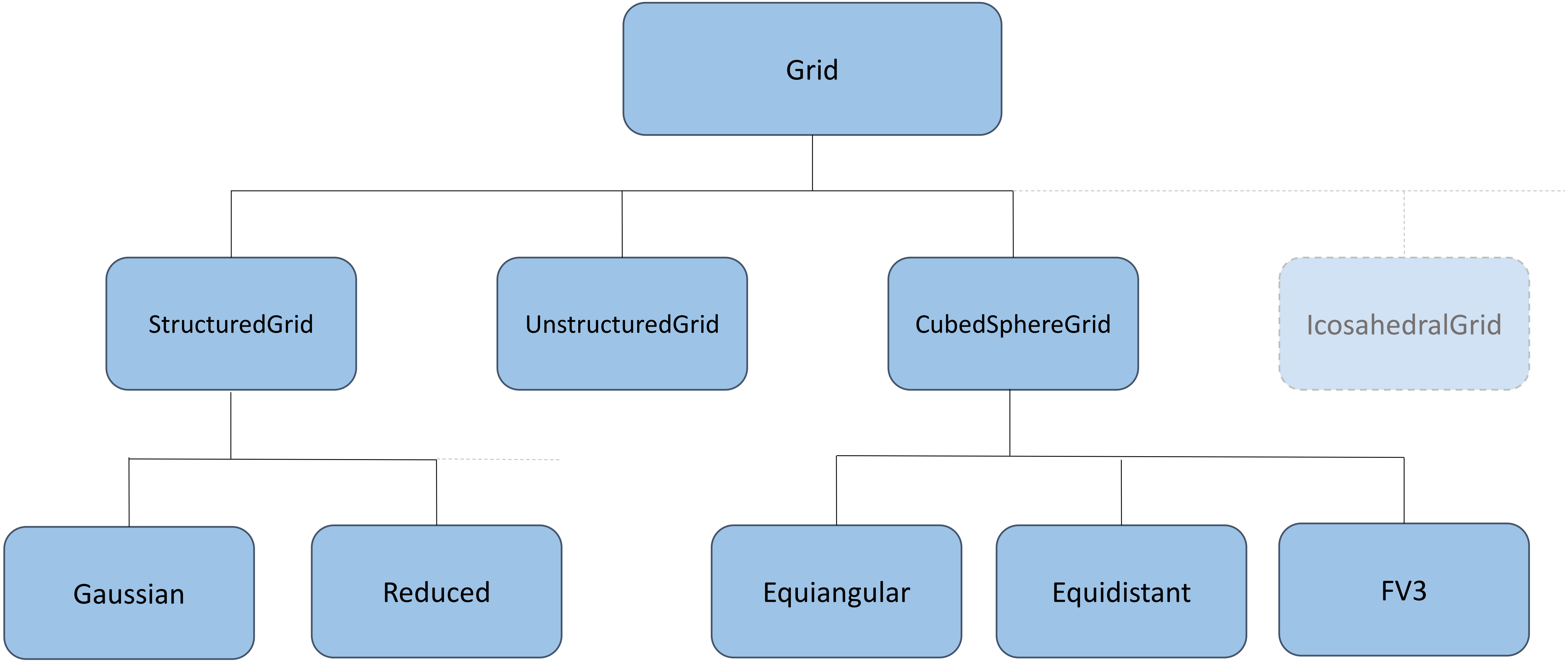
- For FV3/MOM6 we could reduce/eliminate dependency on the FMS library.

- Atlas abstracts away domain specific language for future scalability without code impacts.

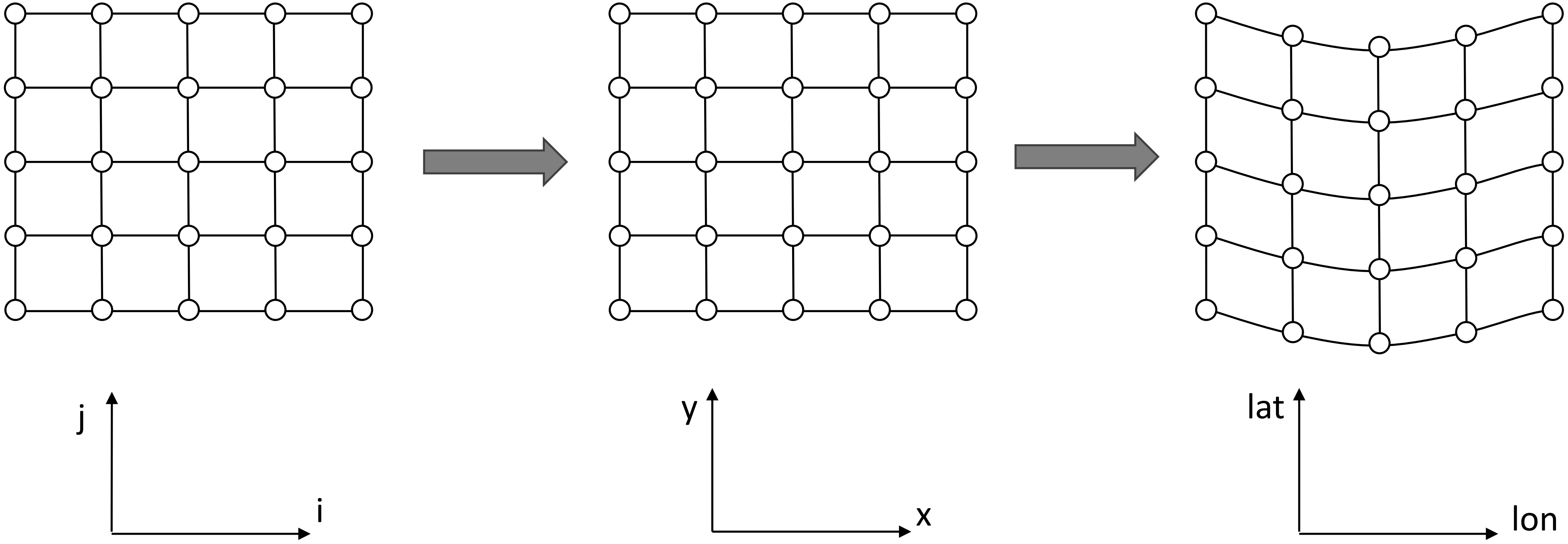
- We can make variable changes considerably more abstract without sacrificing efficiency.

- Unified interpolation.

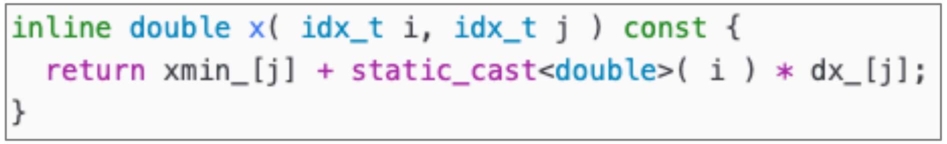
Design


Grid
StructuredGrid
UnstructuredGrid
CubedSphereGrid
IcosahedralGrid
Gaussian
Reduced
Equiangular
Equidistant
FV3

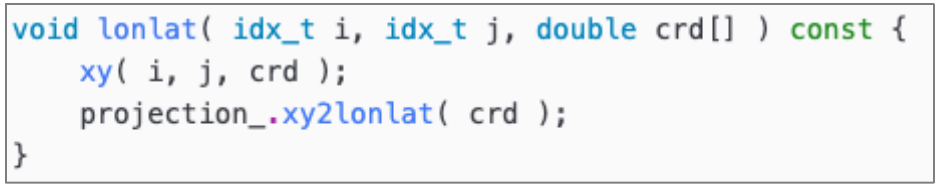
Why not another StructuredGrid?



Structured.h



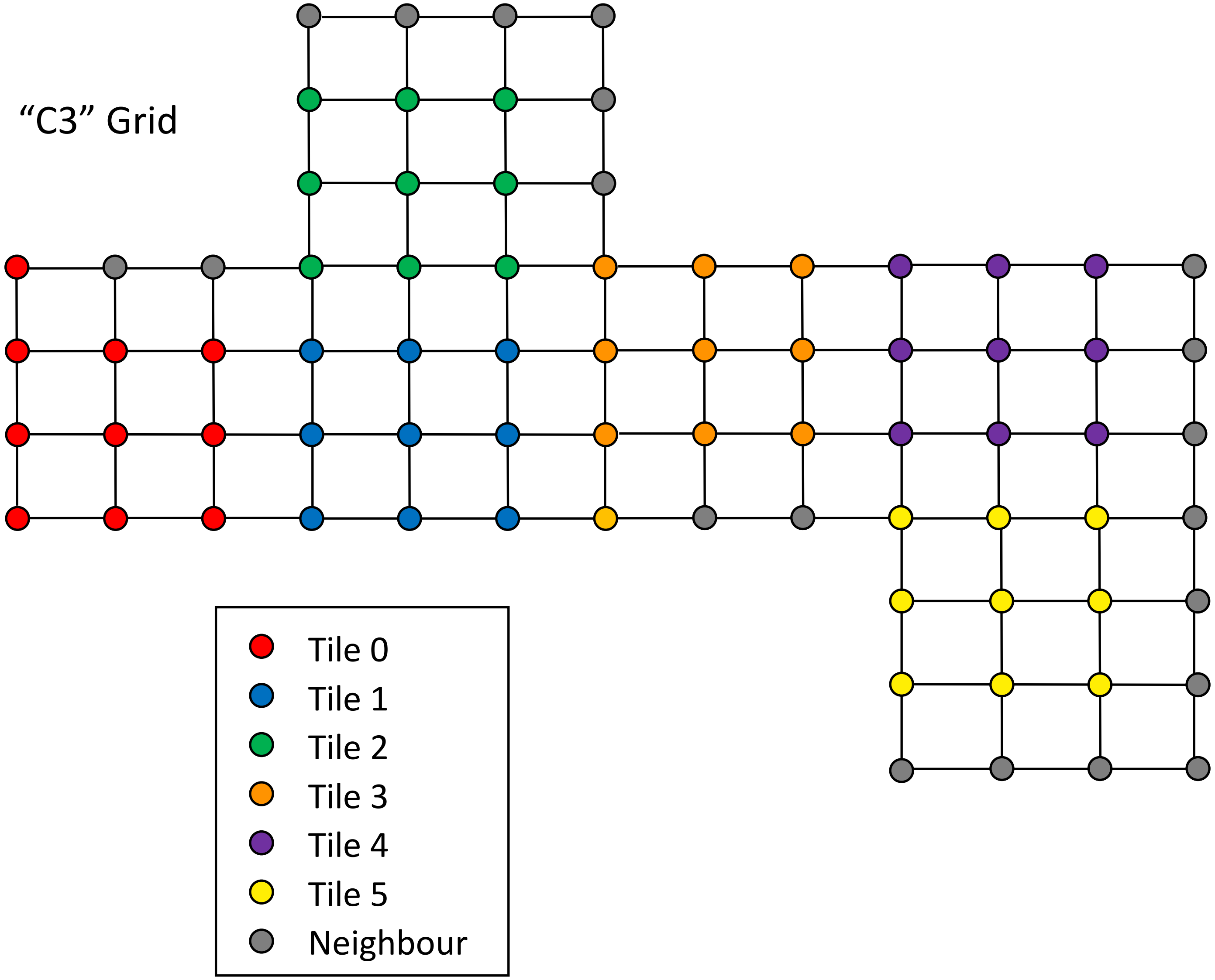
```
inline double x( idx_t i, idx_t j ) const {
  return xmin_[j] + static_cast<double>( i ) * dx_[j];
}
```



```
void lonlat( idx_t i, idx_t j, double crd[] ) const {
    xy( i, j, crd );
    projection_.xy2lonlat( crd );
}
```

For the cubed sphere there is not a direct link between i-j and x-y space. Or between 2D x-y and lon-lat. Also need to have the projection for an entire tile in memory, unlike for the StructuredGrid.

CubedSphereGrid Class

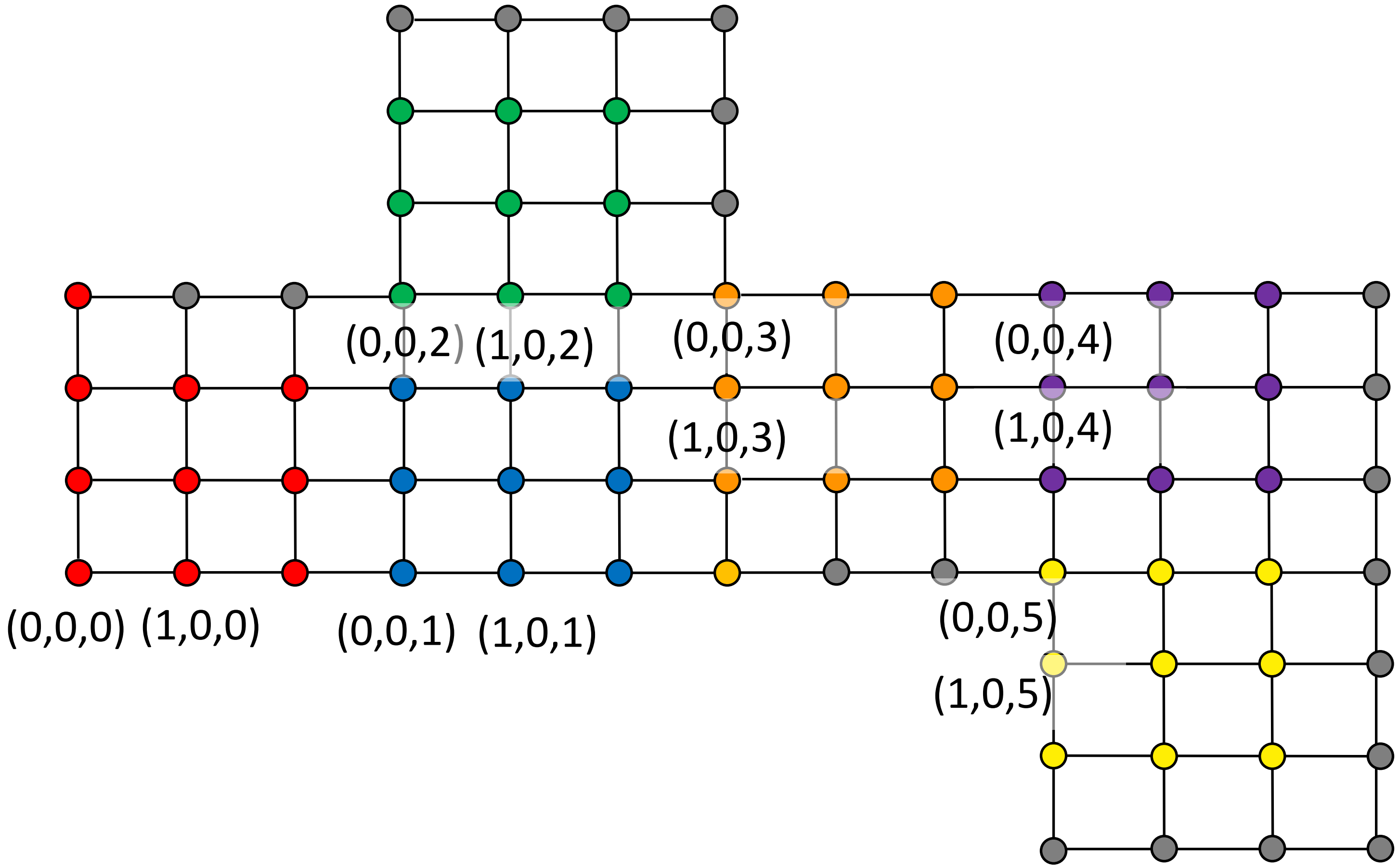


For the CubedSphereGrid class we introduce the additional integer *t*, representing the tile.

For the cubed sphere grid each tile has N^2 grid points, except tile 0 and 3, which have N^2+1 grid points.

In order to fit together with the same number of points on each tile, tiles 3-5 are rotated versions of tiles 0-2.

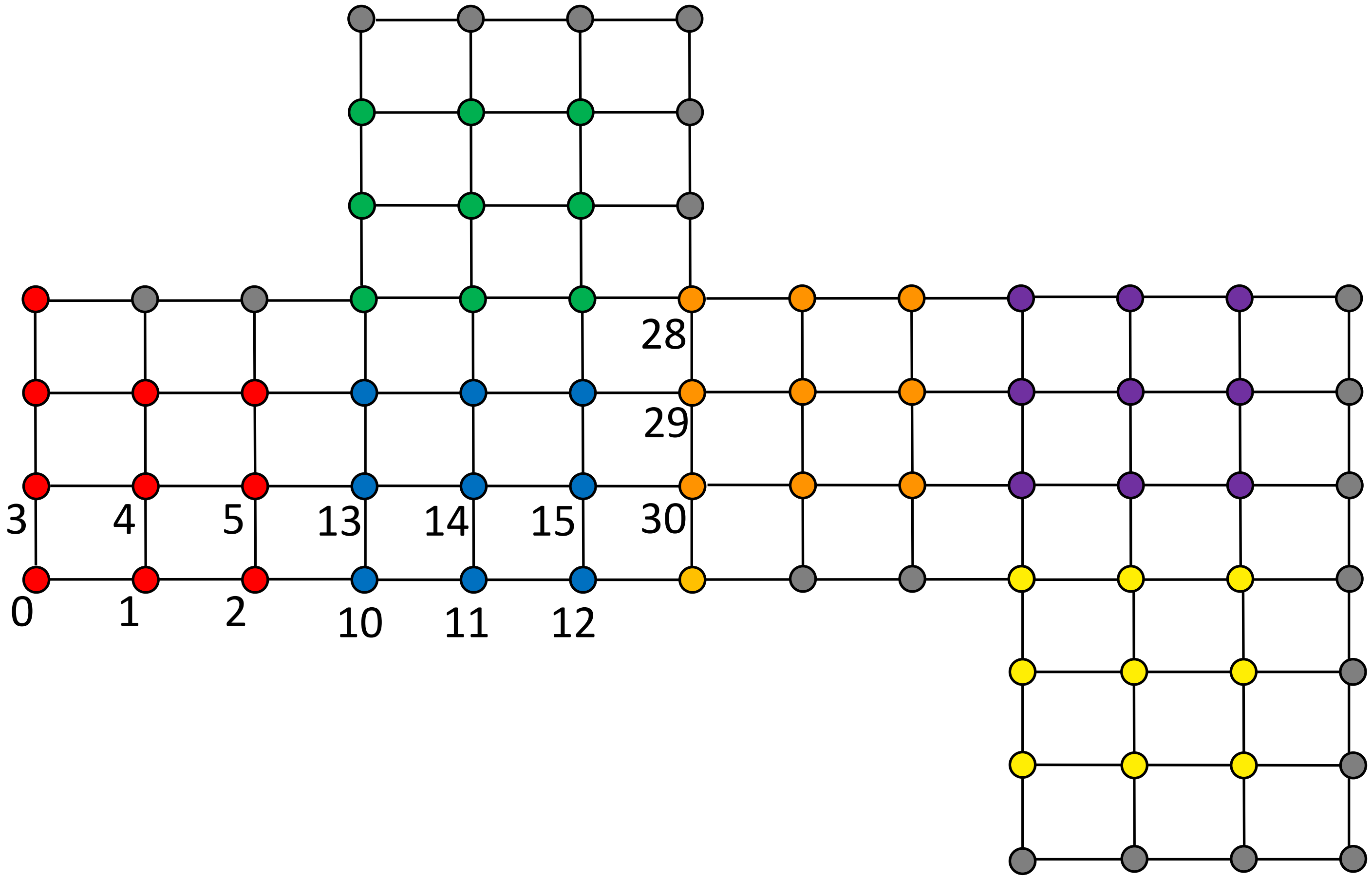
CubedSphereGrid ijt space



- For tiles 0-2 (0,0) is in the lower left-hand corner and i increases from left to right.

- For tiles 3-5 (0,0) is in the upper left-hand corner and i increases from top to bottom.

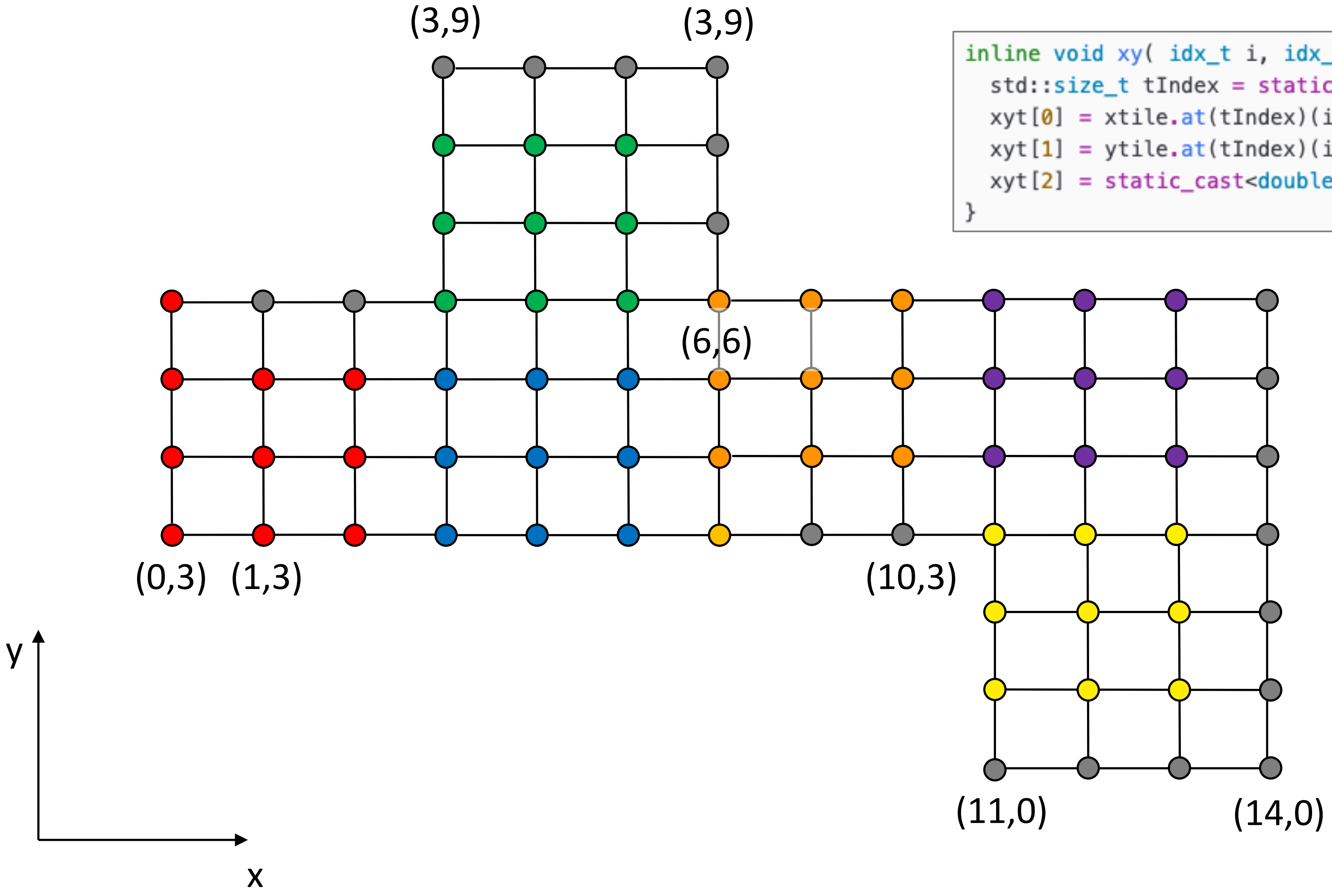
CubedSphereGrid iterator

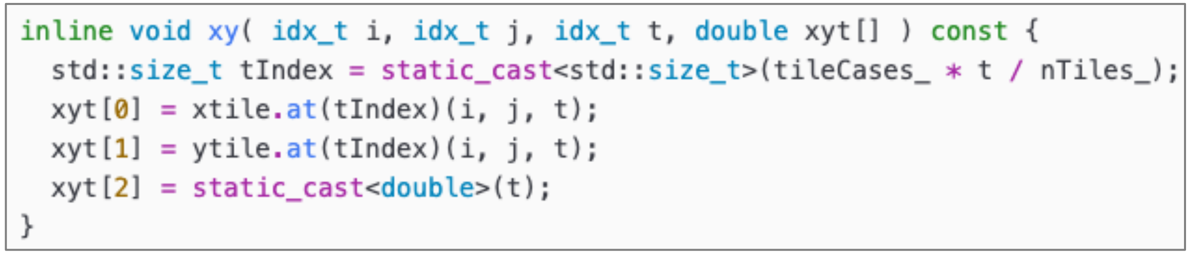


Atlas requires a cubed sphere grid iterator. For the cubed sphere grid it iterates through each tile in the same way that a structured grid would.

There's a little bit of extra complexity due to the extra two grid points and the rotation.

CubedSphereGrid Class X-Y Space





```
inline void xy( idx_t i, idx_t j, idx_t t, double xyt[] ) const {
  std::size_t tIndex = static_cast<std::size_t>(tileCases_ * t / nTiles_);
  xyt[0] = xtile.at(tIndex)(i, j, t);
  xyt[1] = ytile.at(tIndex)(i, j, t);
  xyt[2] = static_cast<double>(t);
}
```

For the x-y space there is no fixed dx and dy since the grid is not regular, so we just picked i-j space without the tile and without the rotation.

Currently the x-y space is not used except to plot the mesh in x-y space.

If there was a proper projection from xy-tile to lonlat this could be improved.

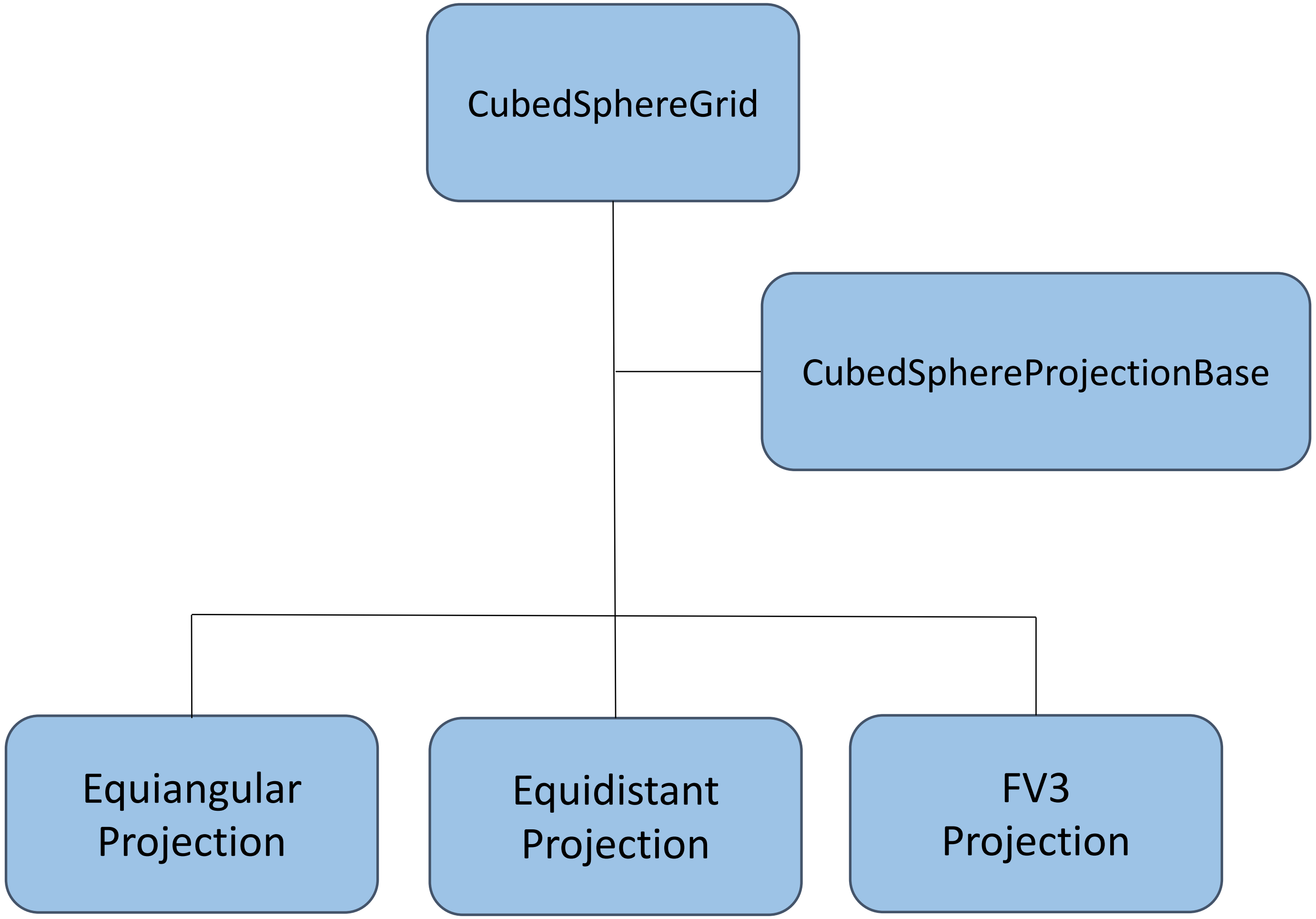
Projections

Through the GridBuilder we have three kinds of CubedSphere grid:

- Equidistant
- Equiangular
- EquidistantFV3

The only difference between these is which projection gets called.

Projections

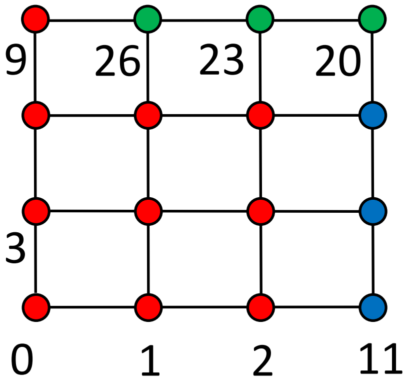


- Creates an Atlas Array to hold the projection for tile 0
- Lambda functions for handling the rotation from Tile 0 to the other 5 tiles.

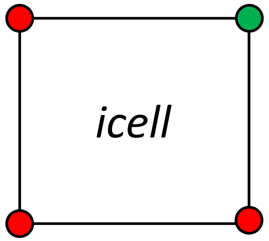
- Constructor creates entire projection for Tile 0.
- xy2lonlat and lonlat2xy just call the base class which calls the lambda functions to return the rotated values.

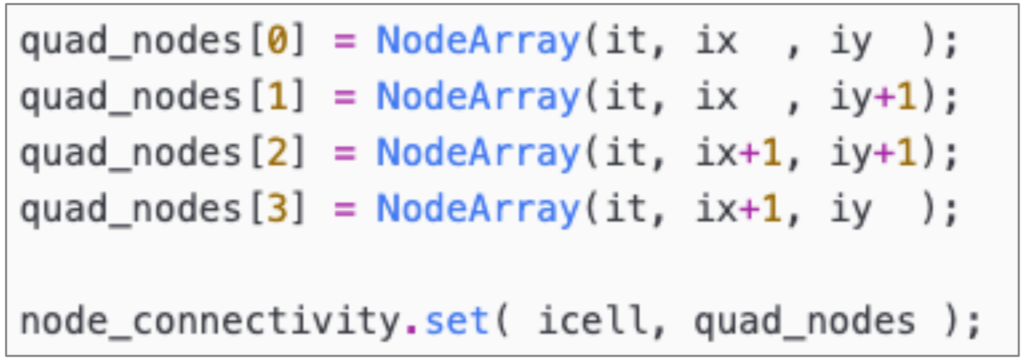
CubedSphereMeshGenerator

Like with the iterator each node of the grid is given an integer value which is stored in NodeArray, which is an Atlas Array object.



NodeArray(0,0,0) = index

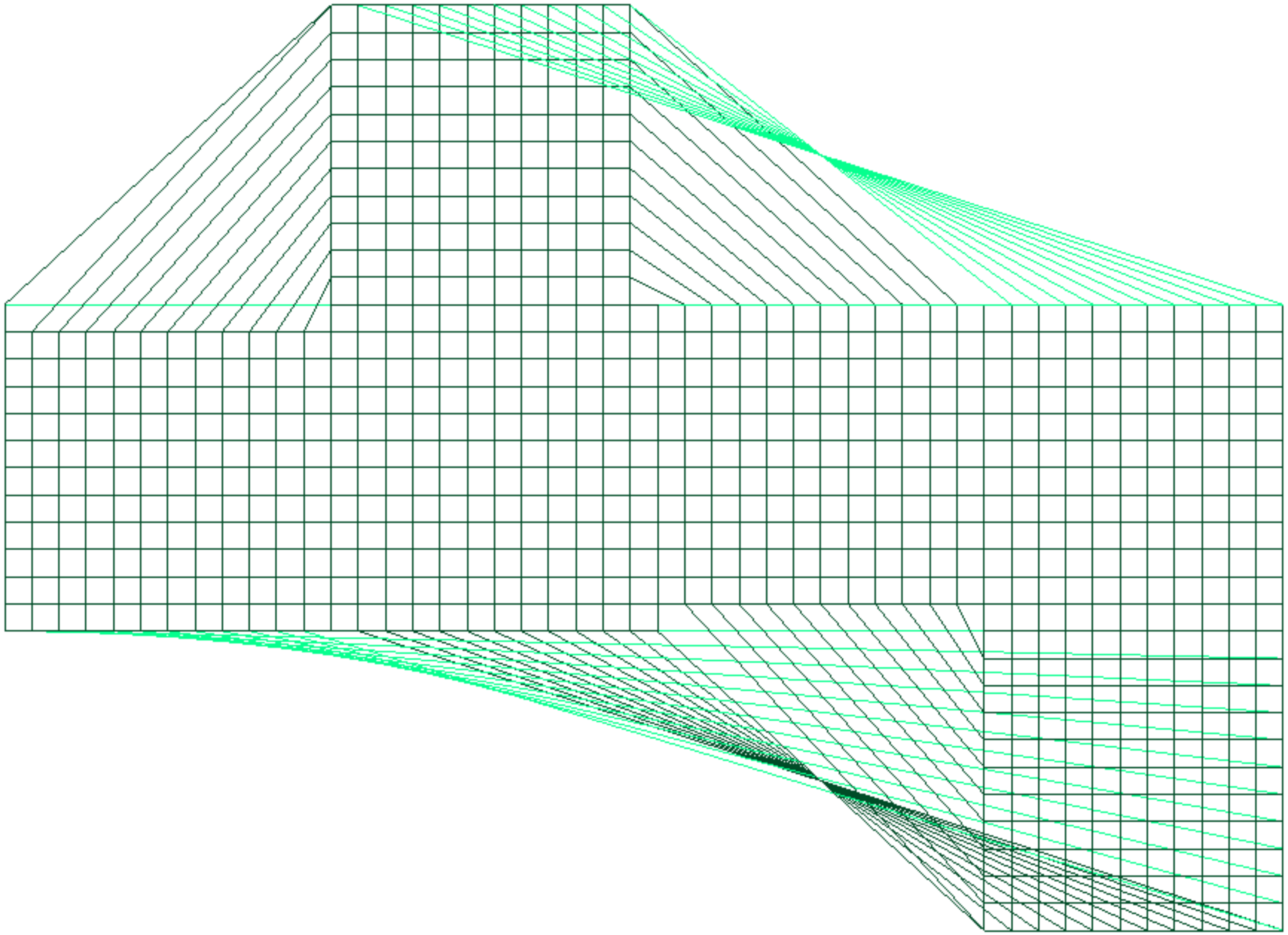




```
quad_nodes[0] = NodeArray(it, ix  , iy  );
quad_nodes[1] = NodeArray(it, ix  , iy+1);
quad_nodes[2] = NodeArray(it, ix+1, iy+1);
quad_nodes[3] = NodeArray(it, ix+1, iy  );

node_connectivity.set( icell, quad_nodes );
```

Mesh (x-y)

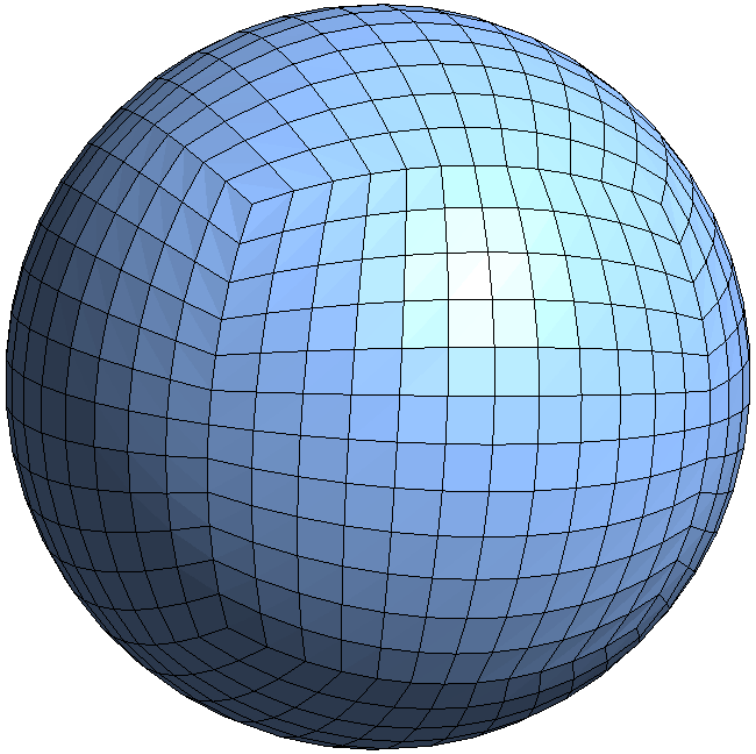


x-y space projection for C12

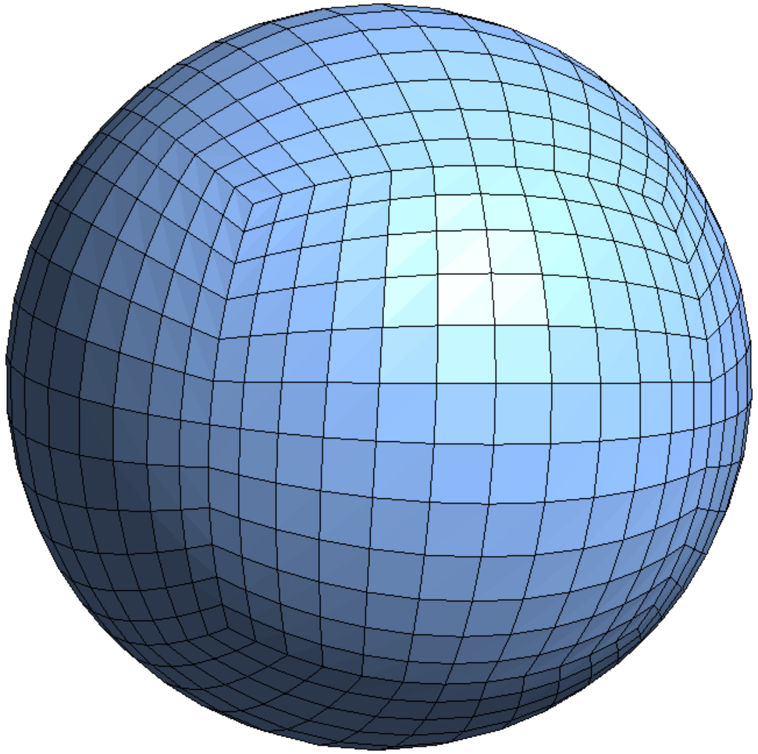
We could also add ghost nodes to the mesh to avoid having the diagonal lines but it’s useful to check the connections all line up correctly.

Front of the mesh lines are shown in dark green, rear is shown in light green.

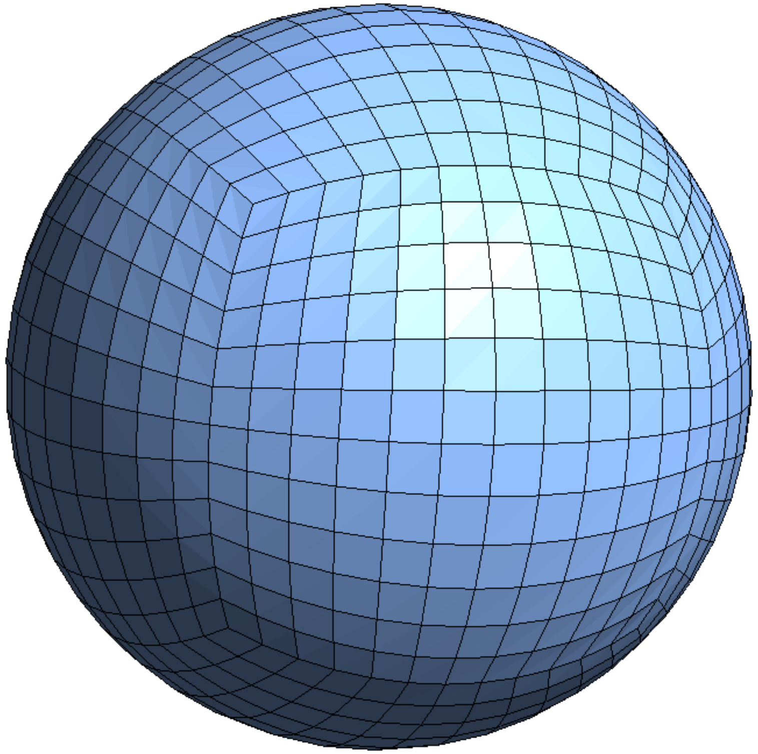
Mesh (lon-lat)



Equiangular (CS-EA-12)
Constant tan(a) in
Cartesian coordinates



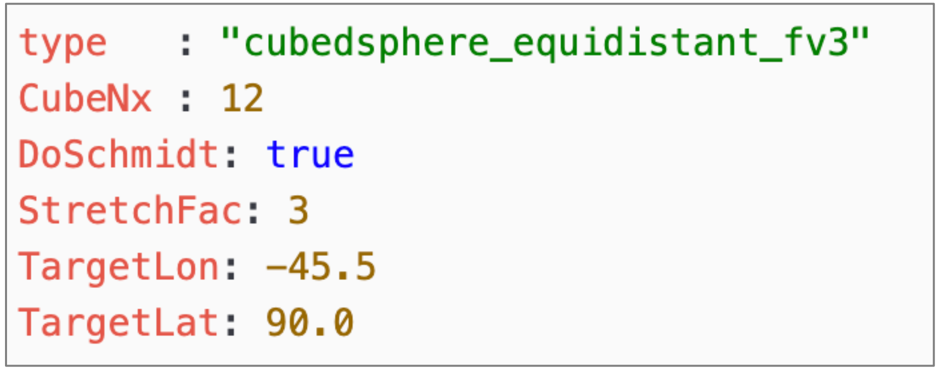
Equidistant (CS-ED-12)
Constant dx,xy in
Cartesian coordinates



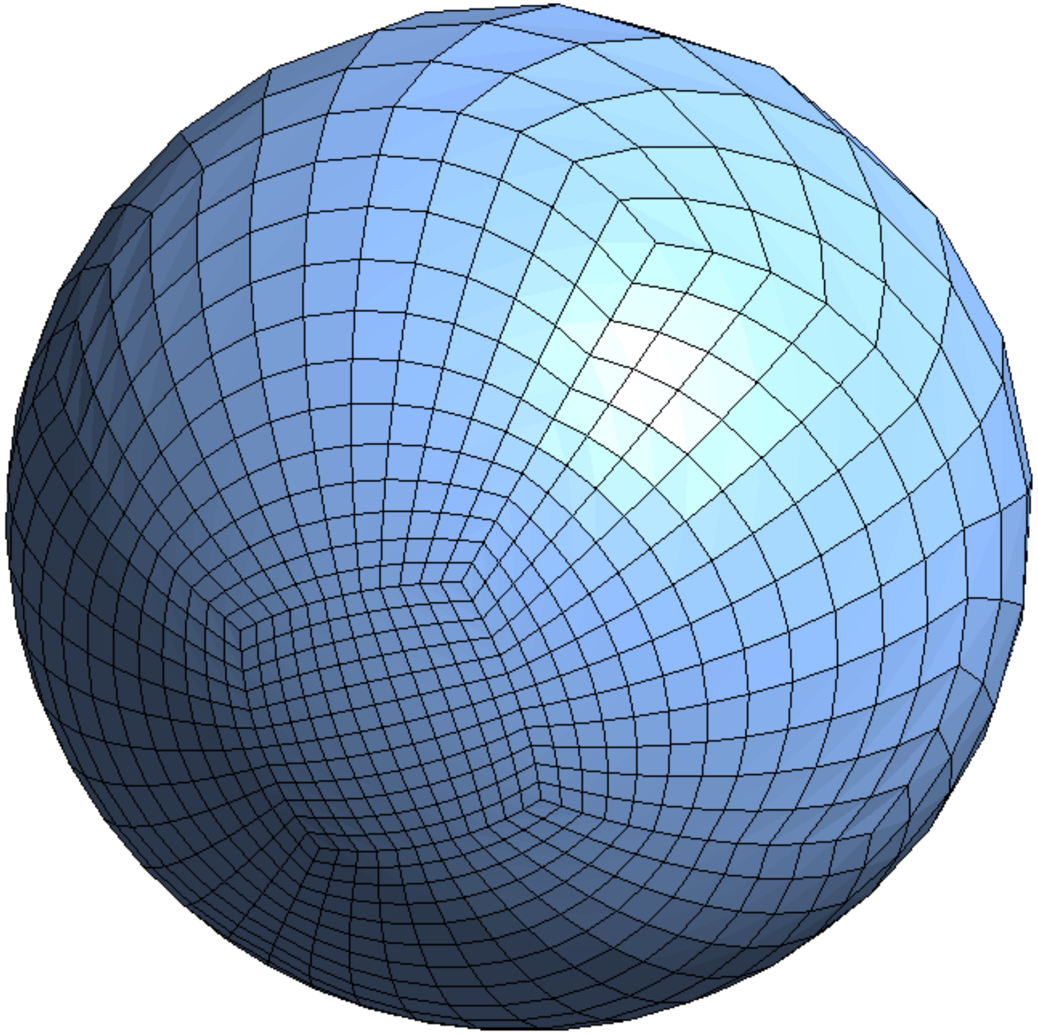
Equidistant FV3 (CS-EDFV3-12)
Equidistant plus some extra
smoothing.

Projection options

- Shift factor for rotating the Cube and moving corners away from land
- Stretch factor and target lon/lat for regional/nested grids.



```
type   : "cubedsphere_equidistant_fv3"
CubeNx : 12
DoSchmidt: true
StretchFac: 3
TargetLon: -45.5
TargetLat: 90.0
```



CS-EDFV3-12

Future work

- Add the Partitioner to allow for domain decomposition and halos. This needs to replicate the partitioning done by FMS to avoid redistribution.
- Add the Fortran API for the different classes.
- Currently it is not possible to have different numbers of grid points in the x versus y direction of each tile and this is required for the new regional applications. Not in FV3 either but would be good to avoid lots of post processing of the grid.
- Document the process of adding a new grid to Atlas for other JCSDA users.

- Convert the fv3-jedi Geometry, State, Increment and GetValues classes to be based on Atlas.
- Create cariable changes for Atlas fields repo.