A Primer for Writing PDE Solvers with Overture

William D. Henshaw
Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551
henshaw@llnl.gov
http://www.llnl.gov/casc/people/henshaw
http://www.llnl.gov/casc/Overture
May 20, 2011

UCRL-MA-132233

Abstract: We describe how to write C++ programs to solve partial differential equations on a single curvilinear grid or on a collection of curvlinear grids that form an overlapping grid. We use classes from the Overture framework to represent grids and grid functions, and to perform operations on the grid functions. Overture makes extensive use of the the serial/parallel array class library A++/P++ to write efficient and portable serial or parallel code.

Contents

1 Introduction

This is a primer for writing C++ codes to solve partial differential equations (PDEs) within the Overture framework. The first set of examples describe how to use Overture to solve problems on a single curvilinear grid. Overture classes are used to represent the grid, to represent the solution on the grid, to operate on the solution (differentiate, for example) and to plot the results.

The second set of examples show how to solve problems on overlapping grids. These examples are very similiar to the single grid case.

The reader is assumed to be familiar with the A++ (parallel) array class library [17]. For the reader wanting to solve problems on overlapping grids, some familiarity with the concept of an overlapping grid is assumed.

In this primer we will introduce and show how to use the following classes

- **Mapping** : A transformation that can be used to represent a curvilinear domain such as a square, annulus, sphere or other more complicated region.
- **MappedGrid** : A logically rectangular grid that is a discrete version of a `Mapping`. A MappedGrid contains grid point coordinates as well as information such as boundary conditions, periodicity, singularities etc.
- **realMappedGridFunction** : A grid function that holds a solution (such as pressure or velocity) on a MappedGrid; this is a glorified A++ array.
- **MappedGridOperators** : the class used with grid functions to define spatial derivatives and to apply boundary conditions.
- **PlotStuff** The class used to interactively plot Overture objects.
- **Ogshow** : A class for saving solutions and other information in a “show file”. A show file can be read by plotStuff (in the Overture/bin directory) to plot solutions.
- **Oges** : The equation solver class that can be used to solve systems of boundary value problems such as Poisson’s equation.
- **CompositeGrid** : An “overlapping composite grid”. Each component grid of a CompositeGrid is a MappedGrid. The grid generator `Ogen`, for example, can generate a CompositeGrid.
- **realCompositeGridFunction** : A grid function that holds a solution (such as the pressure or velocity) on a CompositeGrid.
- **CompositeGridOperators** : classes used with grid collection functions to define spatial derivatives and to apply boundary conditions.
- **Ogen** : The overlapping grid generator that can be used in a moving grid computation to regenerate an overlapping grid when one or more of the component grids change. The grid generator can also be run interactively to create an overlapping grid. See the documentation elsewhere.

These classes are collectively known as “Overture”. “Overture” is an acronym that has absolutely no meaning.

Documentation can be found on the Overture home page, `http://www.llnl.gov/casc/Overture`, and includes the following documents that may be of interest

- A++ Quick Reference Card : `A++P++/DOCS/Quick_Reference_Card.tex`
- A primer for Overture[11].
- Grid and grid function documentation[5].
- Finite difference operators and boundary conditions[4].
- Finite volume operators [1].
- Mapping class documentation [6].
- Show file documentation [9].
- Interactive plotting[10].

- Oges “Equation Solver” documentation [8].
- Interactive grid generation documentation [7].
- The other stuff documentation[12].
- The OverBlown Navier-Stokes flow solver [14][13].

Figure 1 gives an overview of the classes that make up Overture.

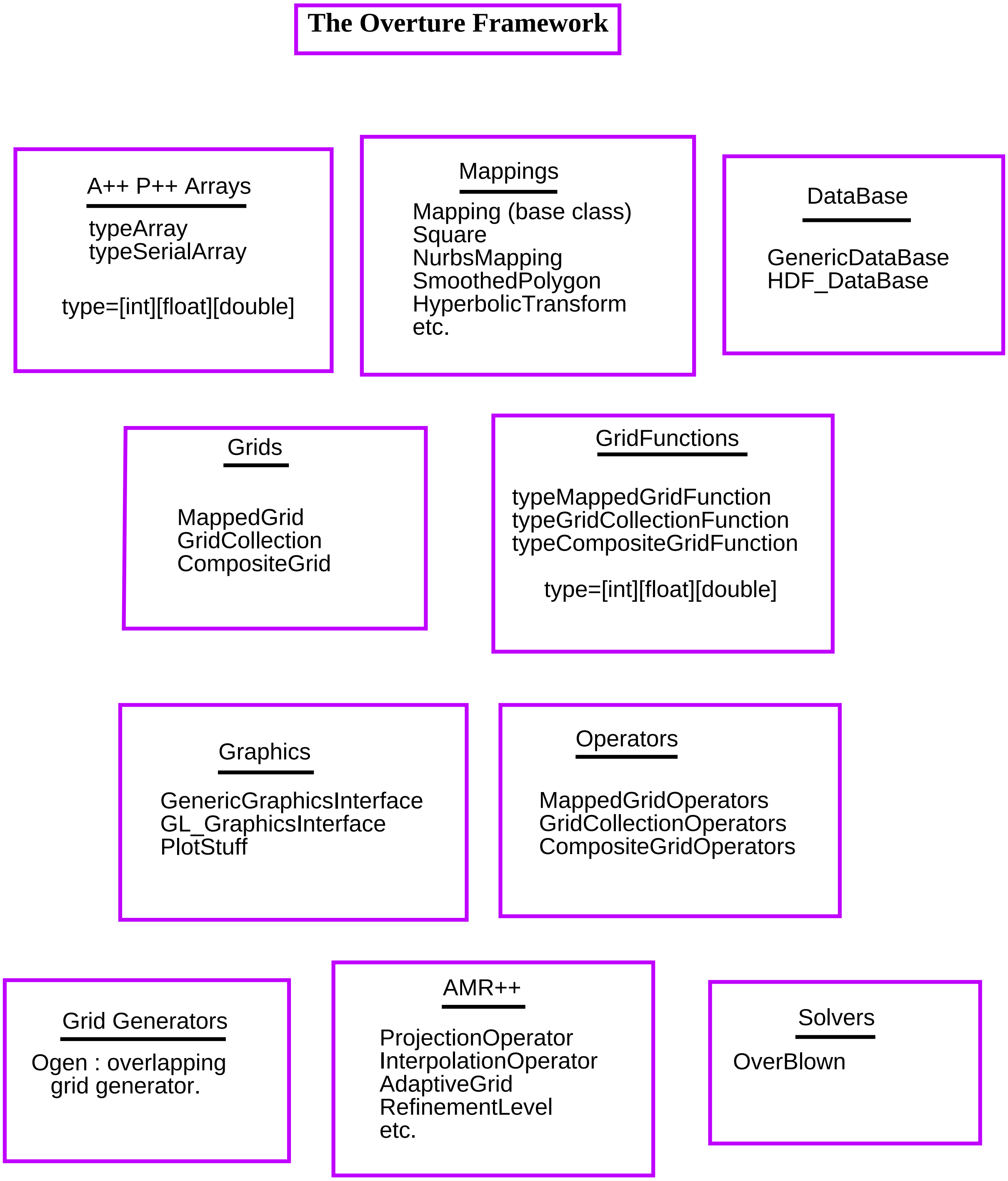


Figure 1: An overview of the Overture classes

2 Getting Started with MappedGrid's

2.1 mappedGridExample1: Mapping's, MappedGrid's, MappedGridFunction's

In this first example we introduce the classes that are used to represent domains (`Mapping`), grids (`MappedGrid`) and and grid functions (`realMappedGridFunction`) (a grid function is the Overture name for a solution variable or a field variable).

In this example the `SquareMapping` is created to define the simple domain that is a square. A `MappedGrid` is created from the `SquareMapping`, the `MappedGrid` is a discrete version of the `Mapping` since it has a specified number of grid points. To define a solution variable on the grid we build a `realMappedGridFunction`. This is an A++ array that is automatically dimensioned to the size of the grid. The grid function is assigned values using array operations (instead of looping over all the grid points, which could also be done).

The `MappedGridOperator` class is used to differentiate the grid function and the `PlotStuff` class is used to interactively display the grid and the grid function.

Notes when you run the code:

- After some information is printed, a PlotStuff window should appear.
- Use the right mouse button to show the popup menu.
- After choosing the `contour` menu item, a contour plot will appear and you will be in the contour plotter with the contour popup menu. The contour is actually a surface which can be seen by rotating the view by clicking, with the left mouse buttom on one of the `x-r`, `x+r`, `y-r`, ... buttons.
- Now choose `erase and exit` to return to the previous menu.
- Use the left mouse button to look at the `file` pull-down menu (in the upper left corner). Here is the command to save a postscript file. Answers to commands, such as the file name for the postscript file are typed on the `Command:` line at the bottom. Any current menu item may also be optionally typed on the command line (one need only type at least as many letters in the name as to distinguish it).

(file `Overture/primer/mappedGridExample1.C`)

```
#include "Overture.h"
#include "PlotStuff.h"
#include "SquareMapping.h"
#include "MappedGridOperators.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------------- \n");
  printf(" Demonstrate mappings, grids, gridfunctions, operators and plotting \n");
  printf(" ------------------------------------------------------------------- \n");

  SquareMapping square(0.,1.,0.,1.);                    // Make a mapping, unit square
  square.setGridDimensions(axis1,11);                   // axis1==0, set no. of grid points
  square.setGridDimensions(axis2,11);                   // axis2==1, set no. of grid points
  MappedGrid mg(square);                                // MappedGrid for a square
  // * mg.changeToAllCellCentered();                         // make a cell centered grid
  mg.update();                                        // This will generate default geometry arrays (e.g. vertex)

  Range all;                                  // a null Range is used as a place-holder below for the coordinates
  realMappedGridFunction u(mg,all,all,all,2);  // create a grid function with 2 components: u(0:10,0:10,0:0,0:1)
  u.setName("Velocity Stuff");                // give names to grid function ...
  u.setName("u Stuff",0);                     // ...and components
  u.setName("v Stuff",1);
  Index I1,I2,I3;

  // mg.dimension()(2,3) : start/end index values for all points on the grid, including ghost-points
  getIndex(mg.dimension(),I1,I2,I3);                         // assign I1,I2,I3 from dimension
  u(I1,I2,I3,0)=sin(Pi*mg.vertex()(I1,I2,I3,axis1))         // component 0 : sin(pi*x)*cos(pi*y)
               *cos(Pi*mg.vertex()(I1,I2,I3,axis2));
  u(I1,I2,I3,1)=cos(Pi*mg.vertex()(I1,I2,I3,axis1))         // component 1 : cos(pi*x)*sin(pi*y)
               *sin(Pi*mg.vertex()(I1,I2,I3,axis2));
```

```
  MappedGridOperators op(mg);                              // operators
  u.setOperators(op);                                      // associate with a grid function
  u.x().display("here is u.x");                            // x derivative
  u.x(all,all,all,0).display("here is u.x(all,all,all,0)");  // x derivative of component 0
  u.x(all,all,all,1).display("here is u.x(all,all,all,1)");  // x derivative of component 1

  getIndex(mg.gridIndexRange(),I1,I2,I3);                    // interior and boundary points
  // compute the error in component 0 of u.x, the notation u.x(I1,I2,I3,0) means only evaluate
  // the derivative for component 0 and at the points (I1,I2,I3) (done for efficiency only)
  real error = max(abs( u.x(I1,I2,I3,0)(I1,I2,I3,0)-
            Pi*cos(Pi*mg.vertex()(I1,I2,I3,axis1))*cos(Pi*mg.vertex()(I1,I2,I3,axis2)) ));
  cout << "Maximum error in component 0 of u.x = " << error << endl;

  bool openGraphicsWindow=TRUE;
  PlotStuff ps(openGraphicsWindow,"mappedGridExample1");      // create a PlotStuff object
  PlotStuffParameters psp;                      // This object is used to change plotting parameters

  aString answer;
  aString menu[] = {                             // Make some menu items
                    "!mappedGridExample1",      // title
                    "contour",
                    "stream lines",
                    "grid",
                    "read command file",
                    "save command file",
                    "erase",
                    "help",
                    "exit",
                    "" };                       // empty string denotes the end of the menu
  for(;;)
  {
    ps.getMenuItem(menu,answer);                // put up a menu and wait for a response
    if( answer=="contour" )
    {
      psp.set(GI_TOP_LABEL,"My Contour Plot");  // set title
      PlotIt::contour(ps,u,psp);                        // contour/surface plots
    }
    else if( answer=="grid" )
      PlotIt::plot(ps,mg,psp);                          // plot the grid
    else if( answer=="stream lines" )
      PlotIt::streamLines(ps,u);                        // streamlines
    else if( answer=="read command file" )
      ps.readCommandFile();
    else if( answer=="save command file" )
      ps.saveCommandFile();
    else if( answer=="erase" )
      ps.erase();
    else if( answer=="help" )
      helpOverture("PR","mappedGridExample1"); // open web page at documentation
    else if( answer=="exit" )
      break;
  }

  Overture::finish();
  return 0;
}
```

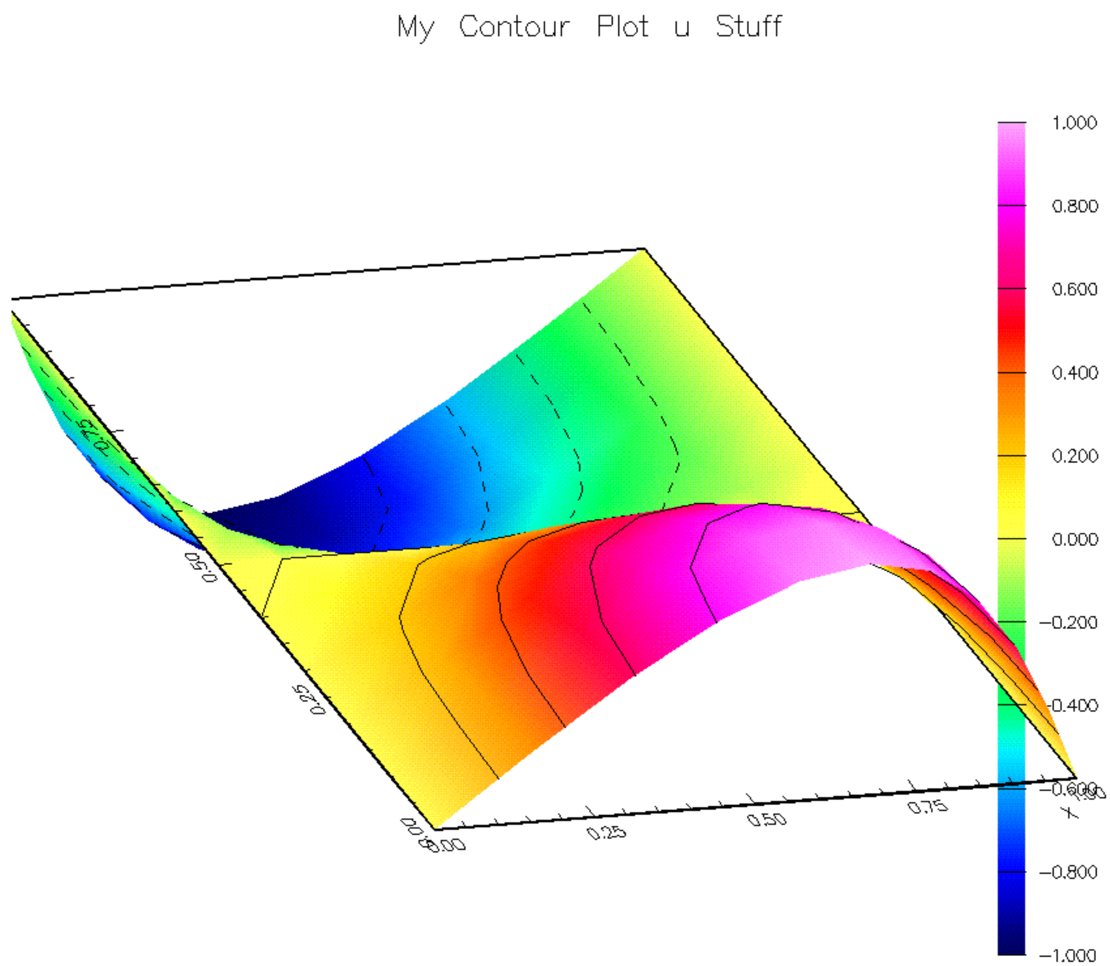


Figure 2: Results from running mappedGridExample1 (contour option, rotated)

2.2 mappedGridExample2: Solve a PDE on an Annulus using operators

In this second example we put together all the classes that were introduced in the first example to solve a simple PDE, a convection diffusion equation:

$$
\begin{aligned}
u_t + a u_x + b u_y &= \nu(u_{xx} + u_{yy}), \\
u(x,y,0) &= 1 \qquad \text{(initial conditions)}, \\
u(x,y,t) &= 0 \qquad \text{on the boundary}.
\end{aligned}
$$

An `AnnulusMapping` is used to create an annulus on which to solve the PDE.

The `MappedGridOperator` class is used to apply boundary conditions, in this case a simple Dirichlet boundary condition (i.e. $u = 0$ given on the boundary). Many other boundary conditions are available.

Notes when you run the code:

- The solution will be plotted every 10 steps.
- Choose the `erase and exit` menu item from the popup menu to continue another 10 steps.

- you can rotate the view (left mouse button click on `x-r` a couple of times) to see the solution surface.

(file `Overture/primer/mappedGridExample2.C`)

```
#include "Overture.h"
#include "PlotStuff.h"
#include "AnnulusMapping.h"
#include "MappedGridOperators.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" -------------------------------------------------------------------- \n");
  printf(" Solve a convection-diffusion equation on an annulus                  \n");
  printf(" Use operators to compute derivatives and apply boundary conditions  \n");
  printf(" Interactively plot results                                           \n");
  printf(" -------------------------------------------------------------------- \n");

  AnnulusMapping annulus;
  annulus.setGridDimensions(axis1,41);               // axis1==0, set no. of grid points
  annulus.setGridDimensions(axis2,13);               // axis2==1, set no. of grid points
  MappedGrid mg(annulus);                            // MappedGrid for a square
  mg.update();                                       // create default variables

  Range all;
  realMappedGridFunction u(mg);
  u.setName("Solution");                          // give names to grid function ...
  u.setName("u",0);                               // ...and components

  Index I1,I2,I3;
  // The A++ array mg.dimension()(2,3) holds index bounds on all points on the grid, including ghost-points
  getIndex(mg.dimension(),I1,I2,I3);               // assign I1,I2,I3 from dimension
  u(I1,I2,I3)=1.;                                 // initial conditions

  MappedGridOperators op(mg);                     // operators
  u.setOperators(op);                             // associate with a grid function

  PlotStuff ps(TRUE,"mappedGridExample2");      // create a PlotStuff object
  PlotStuffParameters psp;         // This object is used to change plotting parameters
  char buffer[80];

  real t=0, dt=.005, a=1., b=1., nu=.1;
  for( int step=0; step<100; step++ )
  {
    if( step % 10 == 0 )
    { // plot contours every 10 steps
      ps.erase();
      psp.set(GI_TOP_LABEL,sPrintF(buffer,"Solution at time t=%e",t));  // set title
      PlotIt::contour(ps, u,psp );
    }

    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) ); // ****** forward Euler time step *****
    t+=dt;
    // apply Boundary conditions
    int component=0;
    u.applyBoundaryCondition(component,BCTypes::dirichlet,BCTypes::allBoundaries,0.);    // set u=0.
    // fix up corners, periodic update:
    u.finishBoundaryConditions();
  }

  Overture::finish();
  return 0;
}
```

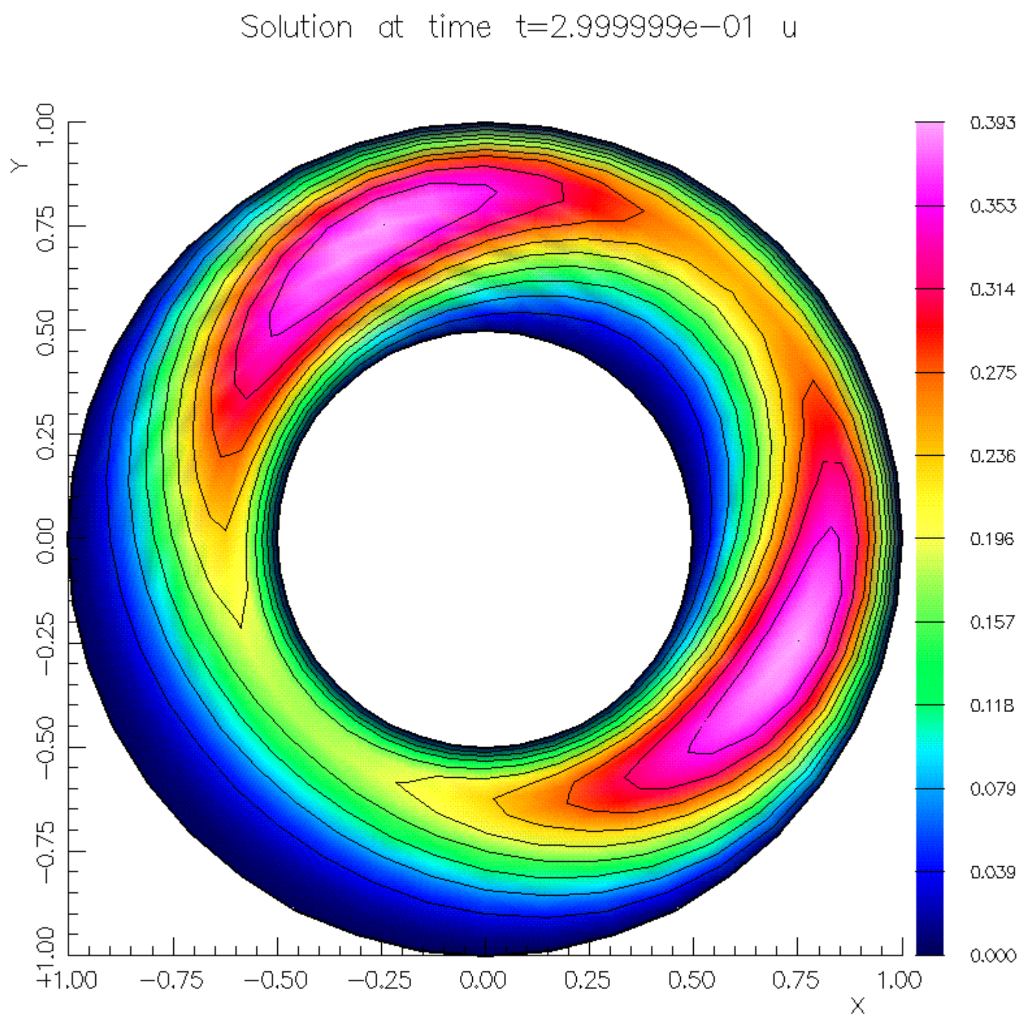


Figure 3: Results from running mappedGridExample2

2.3 mappedGridExample3: Solve a PDE on an Annulus, explicit BC's, use NameList input

In this example we make some minor changes to the previous PDE solver. The `NameList` class is used to interactively enter parameter values by name. This is useful if there are many parameters and only a few need to be changed. The NameList class is described in the PlotStuff documentation.

Boundary conditions are assigned explicitly just to demonstrate how this is done. Using the operators to apply boundary conditions is easier but there may be cases when the predefined boundary conditions don't do what you want to do.

Notes when you run the code:

- When prompted for changes to the parameters you can type "numberOfTimeSteps=100" (or "nts=100") to change the parameter `numberOfTimeSteps`. Type "exit" to continue.
- A contour plot of the initial conditions should appear. Choose "exit" from the popup menu (right mouse button) and the solution will be automatically displayed every 5 time steps ("movie mode").

(file `Overture/primer/mappedGridExample3.C`)

```
#include "Overture.h"
#include "PlotStuff.h"
#include "AnnulusMapping.h"
#include "MappedGridOperators.h"
#include "NameList.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" --------------------------------------------------- \n");
  printf(" Solve a convection-diffusion equation on an annulus \n");
  printf(" Input parameters with the NameList class            \n");
  printf(" Show how to apply boundary conditions explicitly    \n");
  printf(" --------------------------------------------------- \n");

  // Set default values for parameters. These can be optionally changed below
  int numberOfTimeSteps=1000;
  real dt=.01;
  IntegerArray bc(2,3); bc=1;
  NameList nl;       // The NameList object allows one to read in values by name
  aString name(80),answer(80);
  printf(
   " Parameters for Example 3: \n"
   " ------------------------- \n"
   "   name                                                type    default  \n"
   "numberOfTimeSteps  (nts=)                             (int)      %i     \n"
   "time step (dt=)                                       (real)     %f     \n"
   "boundary conditions (bc(side,axis)=)                  (IntegerArray)        \n",
      numberOfTimeSteps,dt);

  // ==========Loop for changing parameters========================
  for( ;; )
  {
    cout << "Enter changes to variables, exit to continue" << endl;
    getLine(answer);
    if( answer=="exit" ) break;
    nl.getVariableName( answer, name );   // parse the answer
    if( name== "numberOfTimeSteps" || name=="nts" )
    {
      numberOfTimeSteps=nl.intValue(answer);
      cout << "numberOfTimeSteps = " << numberOfTimeSteps << endl;
    }
    else if( name== "dt" )
      dt=nl.realValue(answer);
    else if( name== "bc" )
      nl.getIntArray( answer,bc );
    else
      cout << "unknown response: [" << name << "]" << endl;
  }

  Mapping *mapping;                                   // keep a pointer to a mapping
  mapping = new AnnulusMapping();                     // create an Annulus
  mapping->setGridDimensions(axis1,41);               // axis1==0, set no. of grid points
  mapping->setGridDimensions(axis2,13);               // axis2==1, set no. of grid points
  MappedGrid mg(*mapping);                            // MappedGrid for a square
  mg.update();                                        // create default variables

  Range all;
  realMappedGridFunction u;
  u.updateToMatchGrid(mg,all,all,all,1);          // define after declaration (like resize)
  u.setName("Solution");                          // give names to grid function ...
  u.setName("u",0);                               // ...and components

  Index I1,I2,I3, Ib1,Ib2,Ib3;
  // The A++ array mg.dimension()(2,3) holds index bounds on all points on the grid, including ghost-points
  getIndex(mg.dimension(),I1,I2,I3);               // assign I1,I2,I3 from dimension
  u(I1,I2,I3)=1.;                                // initial conditions

  MappedGridOperators op(mg);                    // operators
  u.setOperators(op);                            // associate with a grid function
```

```
  PlotStuff ps(TRUE,"mappedGridExample3");      // create a PlotStuff object
  PlotStuffParameters psp;                       // This object is used to change plotting parameters
  char buffer[80];

  real t=0, a=1., b=1., nu=.05;
  for( int step=0; step<numberOfTimeSteps; step++ )
  {
    if( step % 5 == 0 )
    {
      psp.set(GI_TOP_LABEL,sPrintF(buffer,"Solution at time t=%e",t));  // set title
      ps.erase();
      PlotIt::contour(ps, u,psp );
      psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);     // set this to run in "movie" mode (after first plot)
      ps.redraw(TRUE);
    }
    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) );
    t+=dt;
    // apply Boundary conditions explicitly (just to demonstrate, it is easier to use the operators)
    for( int axis=0; axis<mg.numberOfDimensions(); axis++ )
      for( int side=Start; side<=End; side++ )
      { // only assign BC's on sides with a positive boundary condition:
        if( mg.boundaryCondition()(side,axis) > 0 )
        { // fill in boundary values
          getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
          u(Ib1,Ib2,Ib3)=0.;
        }
      }
    u.periodicUpdate();  // swap periodic edges
  }

  Overture::finish();
  return 0;
}
```

2.4 mappedGridExample4: Write your own Mapping and test it

This example can be skipped for first time users. This program can be used to test a new Mapping if you have written one.

(file `Overture/primer/mappedGridExample4.C`)

```
#include "Overture.h"
#include "PlotStuff.h"
#include "ChannelMapping.h"
#include "MappingInformation.h"
#include "HDF_DataBase.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ---------------------------------------------------------------------------------- \n");
  printf(" Test a user-written Mapping, ChannelMapping.{h,C} \n");
  printf(" Use the checkMapping function to test derivatives, consistency of parameters etc. \n");
  printf(" Save the mapping to a data base and read back again   \n");
  printf(" ---------------------------------------------------------------------------------- \n");

  ChannelMapping channel;

  realArray r(1,2),x(1,2),xr(1,2,2);
  r=.5;
  channel.map(r,x,xr);
  printf(" r=(%f,%f) x=(%f,%f) xr=(%f,%f,%f,%f)\n",r(0,0),r(0,1),x(0,0),x(0,1),
         xr(0,0,0),xr(0,1,0),xr(0,0,1),xr(0,1,1));

  // this function will check the mapping and it's derivatives etc.
  channel.checkMapping();


  // Make interactive changes to the mapping
  PlotStuff ps(TRUE,"mappedGridExample4");      // create a PlotStuff object
  MappingInformation mapInfo;             // parameters used by map.update
  mapInfo.graphXInterface=&ps;            // pass graphics interface
  channel.update(mapInfo);

  // Save the mapping in a data-base file
  HDF_DataBase dataBase;
  cout << "Mount a new database file...\n";
  dataBase.mount("map.dat","I");          // Initialize a database file

  channel.put(dataBase,"my-channel");
  dataBase.unmount();

  // now mount the data-base and read in the mapping
  cout << "Mount an old data base file and read a mapping from it...\n";
  dataBase.mount("map.dat","R");  // mount a data base read-only
  ChannelMapping channel2;
  channel2.get(dataBase,"my-channel");


  r=1.;
  channel2.map(r,x,xr);
  printf(" r=(%f,%f) x=(%f,%f) xr=(%f,%f,%f,%f)\n",r(0,0),r(0,1),x(0,0),x(0,1),
         xr(0,0,0),xr(0,1,0),xr(0,0,1),xr(0,1,1));

  Overture::finish();
  return 0;
}
```

2.5 mappedGridExample5: Generate exact solutions in the Twilight Zone

In this example we show how to use the **method of analytic solutions** to generate an exact solution to a PDE (the so-called twilight-zone solution). Given a PDE that we want to solve, such as,

$$
\begin{aligned}
u_t + a u_x + b u_y &= \nu(u_{xx} + u_{yy}) + f(x,y,t), \\
u(x,y,t) &= g(x,y,t) \qquad \text{on the boundary,}
\end{aligned}
$$

we can choose the forcing function $f(x,y,t)$ and boundary condition $g(x,y,t)$ so that the solution is known. Suppose that we want the solution to be the known function $U(x,y,t)$. For example U may be the polynomial

$$
U = (1 + x + x^2)(1 + y + y^2)(1 + t + t^2).
$$

By choosing

$$
\begin{aligned}
f(x,y,t) &= U_t + aU_x + bU_y - \nu(U_{xx} + U_{yy}) \\
g(x,y,t) &= U(x,y,t)
\end{aligned}
$$

the exact solution for $u(x,y,t)$ will then be $U(x,y,t)$.

The classes `OGPolyFunction` and `OGTrigFunction` have been written to define polynomial and trigonometric functions and their derivatives that can be used to define the known function $U(x,y,z,t)$.

The boundary conditions defined in the operators have been written to optionally use this method of analytic solutions. The operators need to be given the appropriate true solution and they need to be told to force the boundary conditions as shown in the example code below.

When using the polynomial exact solution (default) in the example below the errors in the computed solution are 'zero' (to roundoff, 2×10^{-7} in IEEE single precision).

When using the trigonometric exact solution (by choosing "f=2") the errors at time $t = .5$ are 3.9×10^{-3} in IEEE single precision.

(file `Overture/primer/mappedGridExample5.C`)

```
#include "Overture.h"
#include "PlotStuff.h"
#include "SquareMapping.h"
#include "AnnulusMapping.h"
#include "MappedGridOperators.h"
#include "NameList.h"
#include "OGTrigFunction.h"
#include "OGPolyFunction.h"

enum forcingTypes
{
  noForcing=0,
  poly,
  trig
};

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------------------------------------ \n");
  printf(" Solve a convection-diffusion equation on an annulus                                        \n");
  printf(" Use the method of analytic solutions to obtain an exact solution                           \n");
  printf(" Use OGPolyFunction and OGTrigFunction to define the exact solution and it's derivatives \n");
  printf(" ------------------------------------------------------------------------------------------ \n");

  // Set default values for parameters. These can be optionally changed below
  int numberOfTimeSteps=100;
  real dt=.005;
  IntegerArray bc(2,3); bc=1;
  IntegerArray gridPoints(3); gridPoints=-1;
  int mapType=0;   // 0=square, 1=annulus
  forcingTypes forcingOption=poly;
  int plotOption=TRUE;

  // The NameList object allows one to read in values by name
```

```
NameList nl;
aString name(80),answer(80);
printf(
 " Parameters for Example 5: \n"
 " ------------------------- \n"
 "   name                                                 type    default  \n"
 "numberOfTimeSteps  (nts=)                              (int)      %i      \n"
 "mapType (mt= 0:square, 1=annulus)                      (int)      %i      \n"
 "forcingOption (f= 0:none, 1=poly, 2=trig)              (int)      %i      \n"
 "plotOption (p = 1:on, 0:off)                           (int)      %i      \n"
 "time step (dt=)                                        (real)     %f      \n"
 "gridPoints(axis) (gp(axis)=no. of grid points)         (IntegerArray)        \n"
 "boundary conditions (bc(side,axis)=)                   (IntegerArray)        \n",
    numberOfTimeSteps,mapType,forcingOption,plotOption,dt);

// ==========Loop for changing parameters========================
for( ;; )
{
  cout << "Enter changes to variables, exit to continue" << endl;
  getLine(answer);
  if( answer=="exit" ) break;
  nl.getVariableName( answer, name );   // parse the answer
  if( name== "numberOfTimeSteps" || name=="nts" )
    numberOfTimeSteps=nl.intValue(answer);
  else if( name== "dt" )
    dt=nl.realValue(answer);
  else if( name== "mapType" || name=="mt" )
    mapType=nl.intValue(answer);
  else if( name== "forcingOption" || name=="f" )
    forcingOption=(forcingTypes)nl.intValue(answer);
  else if( name== "plotOption" || name=="p" )
    plotOption=nl.intValue(answer);
  else if( name== "bc" )
    nl.getIntArray( answer,bc );
  else if( name== "gridPoints" || name=="gp")
    nl.getIntArray( answer,gridPoints );
  else
    cout << "unknown response: [" << name << "]" << endl;
}

Mapping *mapping;                                  // keep a pointer to a mapping
if( mapType==0 )
{
  mapping = new SquareMapping();                     // create a Square
  mapping->setGridDimensions(axis1,11);              // axis1==0, set no. of grid points
  mapping->setGridDimensions(axis2,11);              // axis2==1, set no. of grid points
}
else
{
  mapping = new AnnulusMapping();                    // create an Annulus
  mapping->setGridDimensions(axis1,41);              // axis1==0, set no. of grid points
  mapping->setGridDimensions(axis2,13);              // axis2==1, set no. of grid points
}
for( int axis=0; axis<mapping->getDomainDimension(); axis++ )
{
  if( gridPoints(axis)>0 )
    mapping->setGridDimensions(axis,gridPoints(axis));
}
MappedGrid mg(*mapping);                           // MappedGrid for a square
mg.update();                                       // create default variables

Range all;
realMappedGridFunction u(mg);
u.setName("Solution");                          // give names to grid function ...
u.setName("u",0);                               // ...and components

OGFunction *exactPointer;
if( forcingOption==poly )
{
  int degreeOfSpacePolynomial = 2;
  int degreeOfTimePolynomial = 1;
  int nComp = 1;
```

```
    exactPointer = new OGPolyFunction(degreeOfSpacePolynomial,mg.numberOfDimensions(),nComp,
                                      degreeOfTimePolynomial);
  }
  else if( forcingOption==trig )
  {
    real fx=1., fy = 1., fz = 1., ft=1.;        // note that fz is not used in 2D
    //  defines cos(pi*x)*cos(pi*y)*cos(pi*z)*cos(pi*t)
    exactPointer = new OGTrigFunction(fx, fy, fz, ft);
  }
  else if( forcingOption!=0 )
  {
    cout << "Unknown forcing option = " << forcingOption << endl;
    forcingOption=noForcing;
  }
  OGFunction & exact = *exactPointer;  // make a reference for readability

  Index I1,I2,I3, Ib1,Ib2,Ib3;
  // mg.dimension()(2,3) : all points on the grid, including ghost-points
  getIndex(mg.dimension(),I1,I2,I3);               // assign I1,I2,I3 from dimension
  realArray & x= mg.vertex();
  if( forcingOption > 0 )
    u(I1,I2,I3)=exact(mg,I1,I2,I3,0,0.);
  else
    u=1.;

  MappedGridOperators op(mg);                     // operators
  u.setOperators(op);                             // associate with a grid function
  // ***** tell the operators to use the method of analytic solutions for BC's *****
  //       if the forcingOption is greater than 0
  if( forcingOption>0 )
  {
    op.setTwilightZoneFlow(TRUE);
    op.setTwilightZoneFlowFunction(exact);
  }

  PlotStuff ps(plotOption,"mappedGridExample5");        // create a PlotStuff object
  PlotStuffParameters psp;         // This object is used to change plotting parameters
  char buffer[80];

  // Index's for boundary and interior points:
  getIndex(mg.gridIndexRange(),I1,I2,I3);
  real t=0, a=1., b=1., nu=.1;
  for( int step=0; step<numberOfTimeSteps; step++ )
  {
    if( plotOption && step % 20 == 0 )
    {
      sprintf(buffer,"Solution at time t=%e",t);
      psp.set(GI_TOP_LABEL,buffer);  // set title
      PlotIt::contour(ps, u,psp );
    }

    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) );
    if( forcingOption > 0 )
    { // **** Here we add on dt[ U_t + a U_x + b U_y - nu( U_xx + U_yy ) ] ******
      u(I1,I2,I3)+=dt*(exact.t(mg,I1,I2,I3,0,t)
             + a*exact.x(mg,I1,I2,I3,0,t) + b*exact.y(mg,I1,I2,I3,0,t)
             - nu*( exact.xx(mg,I1,I2,I3,0,t) + exact.yy(mg,I1,I2,I3,0,t) ) );
    }
    t+=dt;
    // apply Boundary conditions, this will set u=exact if forcingOption>0
    u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.,t);
    // fix up corners, periodic update:
    u.finishBoundaryConditions();

    real error = max(abs( u(I1,I2,I3)-exact(mg,I1,I2,I3,0,t)));
    printf("t=%6.3f error =%e \n",t,error);
  }

  Overture::finish();
  return 0;
}
```

2.6 mappedGridExample6: Time Step determination

In this example we demonstrate how the time step can be accurately determined for a convection diffusion equation

$$u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$$

The function that determines the time step is in file `Overture/primer/getDt.C` while the file `Overture/primer/-mappedGridExample6.C` uses this function. See the document *Time Step Determination for PDEs with Applications to Programs Written with Overture* [15] (available from the Overture home page) for further details.

(file `Overture/primer/getDt.C`)

```
//===============================================================================================
//
// This function is used by mappedGridExample6
//
//===============================================================================================
#include "Overture.h"
#include "MappedGridOperators.h"
#include "ParallelUtility.h"

real
getDt(const real & cfl,
      const real & a,
      const real & b,
      const real & nu,
      MappedGrid & mg,
      MappedGridOperators & op,
      const real alpha0 = -2.,
      const real beta0  = 1. )
//==========================================================================================
// /Description:
//  Determine the time step for the convection diffusion equation in 2D
//      u_t + a u_x + b u_y = nu( u_xx + u_yy )
//  discretized with the mapping method. Scale the maximum allowable time
//  step by the factor cfl. the stability region is assumed to lie within the
//  ellipse (x/alpha0)^2 + (y/beta0)^2 = 1
// /cfl (input): Scale the time step by this factor (cfl=1 should normally be stable)
// /a (input) : coefficient of u_x
// /b (input) : coefficient of u_y
// /nu (input) : coefficient of (u_xx+u_yy)
// /alpha0, beta0 (input) : parameters defining the ellipse for the stability region
// =========================================================================================
{
  real dt=REAL_MAX;
  if( mg.isRectangular() )
  {
    // ***** rectangular grid *****
    real dx[3];
    mg.getDeltaX(dx);

    dt = cfl * pow(
        pow( fabs(a)*(1./(beta0*dx[0]))+fabs(b)*(1./beta0*dx[1]) , 2.)
        +pow( nu *(4./(alpha0*dx[0]*dx[0])+4./(alpha0*dx[1]*dx[1])) , 2.)
              ,-.5);
  }
  else
  {
    // ***** non-rectangular grid *****

    mg.update(MappedGrid::THEinverseVertexDerivative);  // make sure the jacobian derivatives are built
    // define an alias:
    realMappedGridFunction & rxd = mg.inverseVertexDerivative();

    Index I1,I2,I3;
    getIndex( mg.indexRange(),I1,I2,I3); // Get Index's for the interior+boundary points

#ifdef USE_PPP
    /// In parallel, get the serial array local to this processor
    realSerialArray rx; getLocalArrayWithGhostBoundaries(rxd,rx);
    intSerialArray mask; getLocalArrayWithGhostBoundaries(mg.mask(),mask);
    bool ok=ParallelUtility::getLocalArrayBounds(rxd,rx,I1,I2,I3); // get bounds local to this processor
```

```
#else
    realSerialArray & rx = rxd;
    const intSerialArray & mask = mg.mask();
    bool ok=true;
#endif
    const int nd=mg.numberOfDimensions();
    #define MN(m,n) ((m)+nd*(n))
    #define RX(m,n) rx(I1,I2,I3,MN(m,n))
    if( ok ) // there are points on this processor
    {

      realSerialArray a1(I1,I2,I3), b1(I1,I2,I3);

      // Grid spacings on unit square:
      real dr1 = mg.gridSpacing(axis1);
      real dr2 = mg.gridSpacing(axis2);
      if( nu>0. )
      {
        realSerialArray nu11(I1,I2,I3), nu12(I1,I2,I3), nu22(I1,I2,I3);

        op.derivative(MappedGridOperators::xDerivative,rx,nu11,I1,I2,I3,MN(0,0));  // rxx
        // printf("getDt: max err in rxx=%8.2e\n",max(fabs(nu11-rx.x(I1,I2,I3,0,0)(I1,I2,I3,0,0))));
        op.derivative(MappedGridOperators::yDerivative,rx,nu22,I1,I2,I3,MN(0,1));  // ryy
        // printf("getDt: max err in rxy=%8.2e\n",max(fabs(nu22-rx.y(I1,I2,I3,0,1)(I1,I2,I3,0,1))));
        // a1   = a*RX(0,0) + b*RX(0,1) - nu*( nu11+nu22 );
        a1  = a*RX(0,0) + b*RX(0,1);

        op.derivative(MappedGridOperators::xDerivative,rx,nu11,I1,I2,I3,MN(1,0));  // sxx
        // printf("getDt: max err in sxx=%8.2e\n",max(fabs(nu11-rx.x(I1,I2,I3,1,0)(I1,I2,I3,1,0))));
        op.derivative(MappedGridOperators::yDerivative,rx,nu22,I1,I2,I3,MN(1,1));  // syy
        // printf("getDt: max err in syy=%8.2e\n",max(fabs(nu22-rx.y(I1,I2,I3,1,1)(I1,I2,I3,1,1))));
        // b1   = a*RX(1,0) + b*RX(1,1) - nu*( nu11+nu22 );
        b1   = a*RX(1,0) + b*RX(1,1);

        // nu11 = nu*( r1.x*r1.x + r1.y*r1.y )
        // nu12 = nu*( r1.x*r2.x + r1.y*r2.y )*2
        // nu22 = nu*( r2.x*r2.x + r2.y*r2.y )
        nu11 = nu*( RX(0,0)*RX(0,0) + RX(0,1)*RX(0,1) );
        nu12 = nu*( RX(0,0)*RX(1,0) + RX(0,1)*RX(1,1) )*2.;
        nu22 = nu*( RX(1,0)*RX(1,0) + RX(1,1)*RX(1,1) );

        where( mask(I1,I2,I3)>0 ) // *wdh* 070730
        {
          dt = cfl * min(
            pow(
              pow( abs(a1)*(1./(beta0*dr1))+abs(b1)*(1./beta0*dr2) , 2.)
              +pow(   nu11 *(4./(alpha0*dr1*dr1))
                      +abs(nu12)*(1./(alpha0*dr1*dr2))
                      +nu22 *(4./(alpha0*dr2*dr2)) , 2.)
              ,-.5)
            );
        }

      }
      else
      {
        a1   = a*RX(0,0) + b*RX(0,1);
        b1   = a*RX(1,0) + b*RX(1,1);

        where( mask(I1,I2,I3)>0 )// *wdh* 070730
        {
          dt = cfl * min(
            pow(
              pow( abs(a1)*(1./(beta0*dr1))+abs(b1)*(1./beta0*dr2) , 2.)
              ,-.5)
            );
        }

      }
    } // end if ok
  } // end curvilinear grid

```

```
  dt = ParallelUtility::getMinValue(dt);  // min value over all processors

  return dt;
}


real
getDt(const real & cfl,
        const real & a,
        const real & b,
        const real & c,
        const real & nu,
        MappedGrid & mg,
        MappedGridOperators & op,
        const real alpha0 = -2.,
        const real beta0  = 1. )
//=========================================================================================
// /Description:
//  Determine the time step for the convection diffusion equation in ** 3D ** (or 2D)
//      u_t + a u_x + b u_y + c u_z = nu( u_xx + u_yy )
//  discretized with the mapping method. Scale the maximum allowable time
//  step by the factor cfl. the stability region is assumed to lie within the
//  ellipse (x/alpha0)^2 + (y/beta0)^2 = 1
// /cfl (input): Scale the time step by this factor (cfl=1 should normally be stable)
// /a (input) : coefficient of u_x
// /b (input) : coefficient of u_y
// /c (input) : coefficient of u_z
// /nu (input) : coefficient of (u_xx+u_yy+u_zz)
// /alpha0, beta0 (input) : parameters defining the ellipse for the stability region
// ======================================================================================
{
  if( mg.numberOfDimensions()==2 )
    return getDt(cfl,a,b,nu,mg,op,alpha0,beta0);

  real dt=REAL_MAX;
  if( mg.isRectangular() )
  {
    // ***** rectangular grid *****
    real dx[3];
    mg.getDeltaX(dx);

    dt = cfl * pow(
        pow( fabs(a)*(1./(beta0*dx[0])) +fabs(b)*(1./beta0*dx[1]) +fabs(c)*(1./beta0*dx[2]) , 2.)
        +pow( nu *(4./(alpha0*dx[0]*dx[0])+4./(alpha0*dx[1]*dx[1])+4./(alpha0*dx[2]*dx[2])) , 2.)
              ,-.5);
  }
  else
  {
    // ***** non-rectangular grid *****
    mg.update(MappedGrid::THEinverseVertexDerivative);  // make sure the jacobian derivatives are built
    // define an alias:
    realMappedGridFunction & rxd = mg.inverseVertexDerivative();

    const int nd=mg.numberOfDimensions();
    #undef MN
    #define MN(m,n) ((m)+nd*(n))
    #undef RX
    #define RX(m,n) rx(I1,I2,I3,MN(m,n))

    // Get Index's for the interior+boundary points
    Index I1,I2,I3;
    getIndex( mg.indexRange(),I1,I2,I3);
#ifdef USE_PPP
    /// In parallel, get the serial array local to this processor
    realSerialArray rx; getLocalArrayWithGhostBoundaries(rxd,rx);
    intSerialArray mask; getLocalArrayWithGhostBoundaries(mg.mask(),mask);
    bool ok=ParallelUtility::getLocalArrayBounds(rxd,rx,I1,I2,I3); // get bounds local to this processor
#else
    realSerialArray & rx = rxd;
    const intSerialArray & mask = mg.mask();
    bool ok=true;
#endif
```

```
    if( ok )
    {
      // Grid spacings on unit square:
      real dr1 = mg.gridSpacing(axis1);
      real dr2 = mg.gridSpacing(axis2);
      real dr3 = mg.gridSpacing(axis3);

      realSerialArray rxx(I1,I2,I3), ryy(I1,I2,I3), rzz(I1,I2,I3);
      realSerialArray sxx(I1,I2,I3), syy(I1,I2,I3), szz(I1,I2,I3);
      realSerialArray txx(I1,I2,I3), tyy(I1,I2,I3), tzz(I1,I2,I3);

      op.derivative(MappedGridOperators::xDerivative,rx,rxx,I1,I2,I3,MN(0,0));
      op.derivative(MappedGridOperators::yDerivative,rx,ryy,I1,I2,I3,MN(0,1));
      op.derivative(MappedGridOperators::zDerivative,rx,rzz,I1,I2,I3,MN(0,2));

      op.derivative(MappedGridOperators::xDerivative,rx,sxx,I1,I2,I3,MN(1,0));
      op.derivative(MappedGridOperators::yDerivative,rx,syy,I1,I2,I3,MN(1,1));
      op.derivative(MappedGridOperators::zDerivative,rx,szz,I1,I2,I3,MN(1,2));

      op.derivative(MappedGridOperators::xDerivative,rx,txx,I1,I2,I3,MN(2,0));
      op.derivative(MappedGridOperators::yDerivative,rx,tyy,I1,I2,I3,MN(2,1));
      op.derivative(MappedGridOperators::zDerivative,rx,tzz,I1,I2,I3,MN(2,2));


      realSerialArray imLambda,reLambda;
      imLambda=(abs(a*RX(0,0) +
                    b*RX(0,1) +
                    c*RX(0,2)
                    - nu*( rxx+ryy+rzz ) )*(1./(beta0*dr1)) +
                abs(a*RX(1,0) +
                    b*RX(1,1) +
                    c*RX(1,2)
                    - nu*( sxx+syy+szz ) )*(1./(beta0*dr2)) +
                abs(a*RX(2,0) +
                    b*RX(2,1) +
                    c*RX(2,2)
                    - nu*( txx+tyy+tzz ) )*(1./(beta0*dr3)) );

      reLambda=( ( RX(0,0)*RX(0,0)+
                   RX(0,1)*RX(0,1)+
                   RX(0,2)*RX(0,2) )*(nu*4./(alpha0*dr1*dr1)) +
                 ( RX(1,0)*RX(1,0)+
                   RX(1,1)*RX(1,1)+
                   RX(1,2)*RX(1,2) )*(nu*4./(alpha0*dr2*dr2)) +
                 ( RX(2,0)*RX(2,0)+
                   RX(2,1)*RX(2,1)+
                   RX(2,2)*RX(2,2) )*(nu*4./(alpha0*dr3*dr3)) +
                 abs( RX(0,0)*RX(1,0)+
                      RX(0,1)*RX(1,1)+
                      RX(0,2)*RX(1,2) )*(nu*2.*(1./(alpha0*dr1*dr2)))+
                 abs( RX(0,0)*RX(2,0)+
                      RX(0,1)*RX(2,1)+
                      RX(0,2)*RX(2,2) )*(nu*2.*(1./(alpha0*dr1*dr3))) +
                 abs( RX(1,0)*RX(2,0)+
                      RX(1,1)*RX(2,1)+
                      RX(1,2)*RX(2,2) )*(nu*2.*(1./(alpha0*dr2*dr3))) );

      where( mask(I1,I2,I3)>0 )// *wdh* 070730
      {
        dt = cfl * min( pow( imLambda*imLambda + reLambda*reLambda , -.5 ) );
      }

    } // end if ok

  } // end curvilinear grid

  dt = ParallelUtility::getMinValue(dt);  // min value over all processors
  return dt;
}
```

(file `Overture/primer/mappedGridExample6.C`)

```
//===============================================================================================
//
// This example shows how to determine the time step for a 2D convection diffusion equation
//
//  Bill Henshaw
//===============================================================================================
#include "Overture.h"
#include "PlotStuff.h"
#include "SquareMapping.h"
#include "AnnulusMapping.h"
#include "MappedGridOperators.h"
#include "NameList.h"

#define UTRUE(x,y,t) (x)*(1.-(x))*(y)*(1.-(y))*(1.+(t))
#define UTRUEX(x,y,t) (1.-2.*(x))*(y)*(1.-(y))*(1.+(t))
#define UTRUEY(x,y,t) (x)*(1.-(x))*(1.-2.*(y))*(1.+(t))
#define UTRUET(x,y,t) (x)*(1.-(x))*(y)*(1.-(y))
#define UTRUEXX(x,y,t) -2.*(y)*(1.-(y))*(1.+(t))
#define UTRUEYY(x,y,t) -2.*(x)*(1.-(x))*(1.+(t))

#define FORCE(x,y,t) UTRUET(x,y,t)+a*UTRUEX(x,y,t)+b*UTRUEY(x,y,t) \
                     -nu*(UTRUEXX(x,y,t)+UTRUEYY(x,y,t))

real
getDt(const real & cfl,
      const real & a,
      const real & b,
      const real & nu,
      MappedGrid & mg,
      MappedGridOperators & op,
      const real alpha0 = -2.,
      const real beta0  = 1. );

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------------ \n");
  printf(" Solve a convection-diffusion equation on a square or annulus       \n");
  printf(" Determine the correct time step using the getDt function (getDt.C) \n");
  printf(" ------------------------------------------------------------------ \n");

  // Set default values for parameters. These can be optionally changed below
  int numberOfTimeSteps=100;
  real dt=.005, cfl=.5;
  IntegerArray bc(2,3); bc=1;
  IntegerArray gridPoints(3); gridPoints=-1;
  int mapType=0;   // 0=square, 1=annulus
  // The NameList object allows one to read in values by name
  NameList nl;
  aString name(80),answer(80);
  printf(
   " Parameters for Example 3: \n"
   " ------------------------- \n"
   "   name                                              type    default  \n"
   "numberOfTimeSteps  (nts=)                           (int)      %i     \n"
   "mapType (mt= 0:square, 1=annulus)                   (int)      %i     \n"
   "cfl                                                 (real)     %f     \n"
   "time step (dt=)                                     (real)     %f     \n"
   "gridPoints(axis) (gp(axis)=no. of grid points)      (intArray)        \n"
   "boundary conditions (bc(side,axis)=)                (intArray)        \n",
      numberOfTimeSteps,mapType,cfl,dt);

  // =========Loop for changing parameters=======================
  for( ;; )
  {
    cout << "Enter changes to variables, exit to continue" << endl;
    getLine(answer);
    if( answer=="exit" ) break;
```

```
    nl.getVariableName( answer, name );   // parse the answer
    if( name== "numberOfTimeSteps" || name=="nts" )
      numberOfTimeSteps=nl.intValue(answer);
    else if( name== "dt" )
      dt=nl.realValue(answer);
    else if( name== "cfl" )
      cfl=nl.realValue(answer);
    else if( name== "mapType" || name=="mt" )
      mapType=nl.intValue(answer);
    else if( name== "bc" )
      nl.getIntArray( answer,bc );
    else if( name== "gridPoints" || name=="gp")
      nl.getIntArray( answer,gridPoints );
    else
      cout << "unknown response: [" << name << "]" << endl;

  }

  Mapping *mapping;                                  // keep a pointer to a mapping
  if( mapType==0 )
  {
    mapping = new SquareMapping();                     // create a Square
    mapping->setGridDimensions(axis1,11);              // axis1==0, set no. of grid points
    mapping->setGridDimensions(axis2,11);              // axis2==1, set no. of grid points
  }
  else
  {
    mapping = new AnnulusMapping();                    // create an Annulus
    mapping->setGridDimensions(axis1,41);              // axis1==0, set no. of grid points
    mapping->setGridDimensions(axis2,13);              // axis2==1, set no. of grid points
  }
  for( int axis=0; axis<mapping->getDomainDimension(); axis++ )
  {
    if( gridPoints(axis)>0 )
      mapping->setGridDimensions(axis,gridPoints(axis));
  }
  MappedGrid mg(*mapping);                           // MappedGrid for a square
  mg.update();                                       // create default variables

  Range all;
  realMappedGridFunction u(mg);
  u.setName("Solution");                          // give names to grid function ...
  u.setName("u",0);                               // ...and components

  Index I1,I2,I3, Ib1,Ib2,Ib3;
  // mg.dimension()(2,3) : all points on the grid, including ghost-points
  getIndex(mg.dimension(),I1,I2,I3);               // assign I1,I2,I3 from dimension
  realArray & x= mg.vertex();
  u(I1,I2,I3)=UTRUE(x(I1,I2,I3,0),x(I1,I2,I3,1),0.);       // initial conditions

  MappedGridOperators op(mg);                    // operators
  u.setOperators(op);                            // associate with a grid function

  PlotStuff ps(TRUE,"mappedGridExample6");      // create a PlotStuff object
  PlotStuffParameters psp;         // This object is used to change plotting parameters
  char buffer[80];

  real t=0, a=1., b=1., nu=.1;

  // getDt needs the inverseVertexDerivative
  mg.update(MappedGrid::THEinverseVertexDerivative);

  dt = getDt( cfl,a,b,nu,mg,op );
  cout << " dt from getDt = " << dt << endl;

  int tStep=numberOfTimeSteps/10;

  for( int step=0; step<numberOfTimeSteps; step++ )
  {
    if( step % tStep == 0 )
    {
      psp.set(GI_TOP_LABEL,sPrintF(buffer,"Solution at time t=%e",t));  // set title
```

```
      PlotIt::contour(ps, u,psp );
    }

    getIndex(mg.dimension(),I1,I2,I3);
    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) +FORCE(x(I1,I2,I3,0),x(I1,I2,I3,1),t) );
    t+=dt;
    // apply Boundary conditions
    // apply Boundary conditions
    for( int axis=0; axis<mg.numberOfDimensions(); axis++ )
      for( int side=Start; side<=End; side++ )
      { // only assign BC's on sides with a positive boundary condition:
        if( mg.boundaryCondition()(side,axis) > 0 )
        { // fill in boundary values
          getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
          u(Ib1,Ib2,Ib3)=UTRUE(x(Ib1,Ib2,Ib3,0),x(Ib1,Ib2,Ib3,1),t);
        }
      }
    u.periodicUpdate();  // swap periodic edges

    getIndex(mg.gridIndexRange(),I1,I2,I3);
    real error = max(abs( u(I1,I2,I3)-UTRUE(x(I1,I2,I3,0),x(I1,I2,I3,1),t) ));
    cout << "t=" << t << ", error =" << error << endl;
  }

  Overture::finish();
  return 0;
}
```

2.7 Example: Steady state, linearized, incompressible Navier-Stokes Equations

This example shows how to solve a system of equations that looks something like the steady state incompressible Navier-Stokes equations. The system we solve is

$$
\begin{aligned}
\nu\Delta u - (u_0(x,y)u_x + v_0(x,y)u_y + p_x) &= f_0 \\
\nu\Delta v - (u_0(x,y)v_x + v_0(x,y)v_y + p_y) &= f_1 \\
\Delta p - \delta(u_x + v_y) &= f_2
\end{aligned}
$$

with boundary conditions

$$
\begin{aligned}
\text{wall:} &\quad u = \text{given} \quad \text{v} = \text{given}, \quad \text{p}_\text{n} = \text{given} \\
\text{inflow:} &\quad u = \text{given}, \quad \text{v} = \text{given}, \quad \text{p} = \text{given} \\
\text{outflow:} &\quad u_n = \text{given} \quad \text{v}_\text{n} = \text{given}, \quad \text{p} = \text{given}
\end{aligned}
$$

The damping factor δ helps to keep $u_x + v_y$ small, see the papers [3],[16] for more details on discretizing the incompressible Navier-Stokes equations.

The system of equations is built using coefficient matrices, see the operator documentation, available from the Overture home page, for other examples. The program solves the equations on a square with an inflow boundary on the left, an outflow boundary on the right and no-slip walls on the top and bottom. The forcing functions f_0, f_1 and f_2 are chosen so that the exact solution is known. This known solution is a quadratic polynomial for which the method should give the exact answer. Indeed if you run this example you should see that the computed errors are "zero" (round off level).

(file `Overture/primer/lins.C`)

3 Getting Started with CompositeGrid's

A `CompositeGrid` is a class that holds an overlapping grid. An overlapping grid can be created with the interactive grid generation program `ogen`, and saved in a data-base (HDF) file. Application programs such as the examples that follow can easily read the data-base file to create an overlapping grid.

3.1 Example 1: CompositeGrid's and MappedGrid's

Here is an example of how to create a CompositeGrid from a data base file created by the interactive grid generation program `ogen`.

(file `Overture/primer/example1.C`)

```
#include "Overture.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ---------------------------------------------------------- \n");
  printf(" Read an overlapping grid from a data base file             \n");
  printf(" Loop over the component grids and display the boundary     \n");
  printf("   conditions and the grid points (vertex array)            \n");
  printf(" ---------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "Enter the name of the overlapping grid data base file " << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )   // loop over component grids
  {
    cg[grid].boundaryCondition().display("Here are the boundary conditions");
    cg[grid].vertex().display("Here are the vertex coordinates");

    // A Composite grid is a list of MappedGrid's. To save typing we can
    // make a reference (alias):
    MappedGrid & mg = cg[grid];                                  // make a reference to the MappedGrid
    mg.boundaryCondition().display("Here is boundaryCondition again");  // same result as above


    realArray & ci = cg.interpolationCoordinates[grid];
    intArray & il = cg.interpoleeLocation[grid];
    intArray & donor = cg.interpoleeGrid[grid];

    for( int i=0; i<cg.numberOfInterpolationPoints(grid); i++  )
    {
      printF("%i %i %26.18e %26.18e %i %i %i\n",grid,i,ci(i,0),ci(i,1),il(i,0),il(i,1),donor(i));
    }

  }

  Overture::finish();
  return 0;
}

```

We first read in a `CompositeGrid` from the data base file that was created with `ogen`. We then loop over the component grids and print out some variables. A component grid is actually a "MappedGrid", as shown in the example. A `MappedGrid` is so named since it contains a mapping function. See sections 6 and 7 for a brief description of the variables that are contained in a `MappedGrid` and a `CompositeGrid`.

When I run this example the program will prompt for the name of the overlapping grid data base file, and I will enter the name of the file that I created with `ogen`, for example `/home/henshaw.0/Overture/cgsh/square5.hdf`.

The file `Overture/primer/gridQuery.C` is a program that can be used to read in and display various information about a CompositeGrid. .

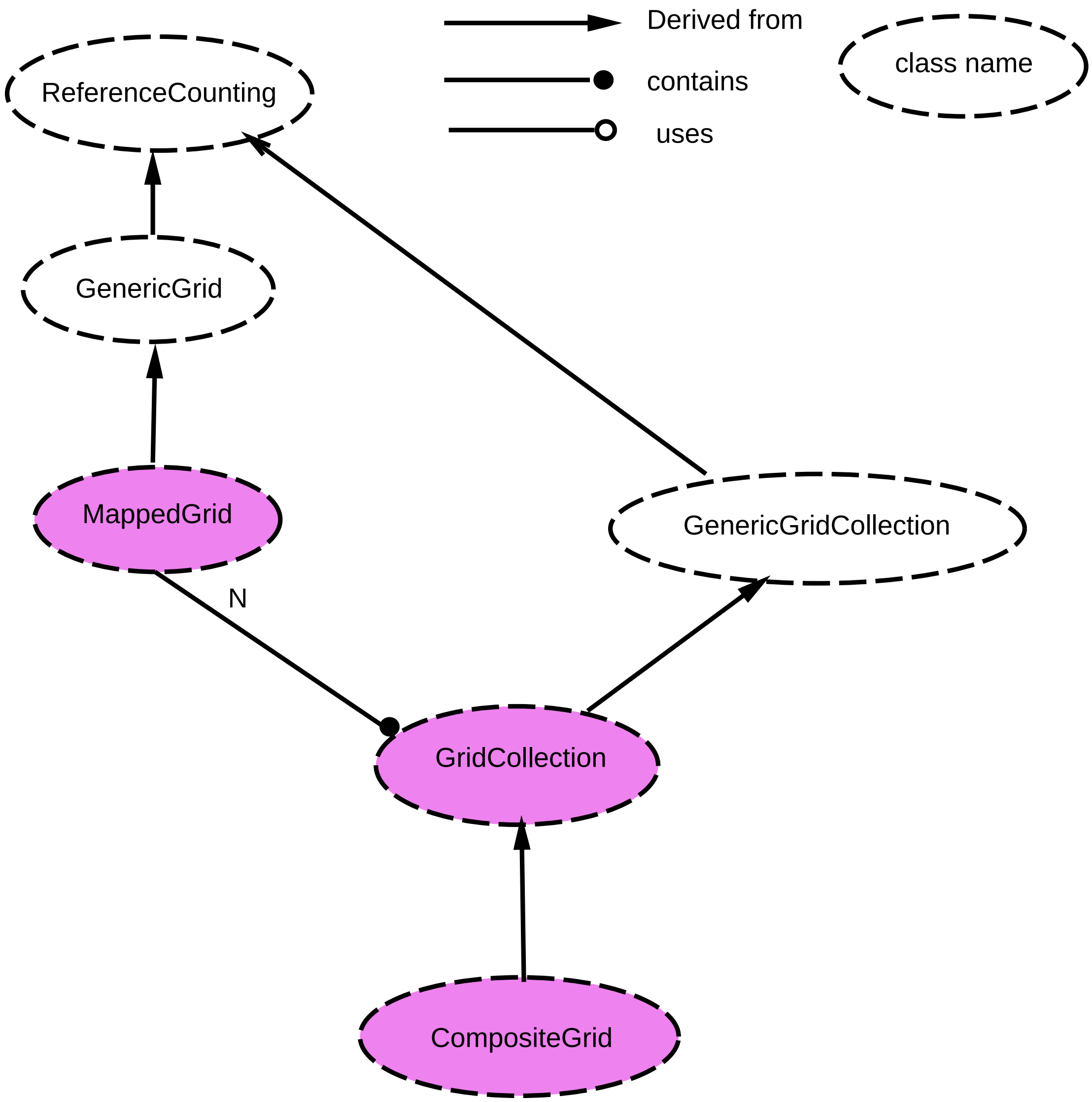


Figure 4: Class diagram for grid classes

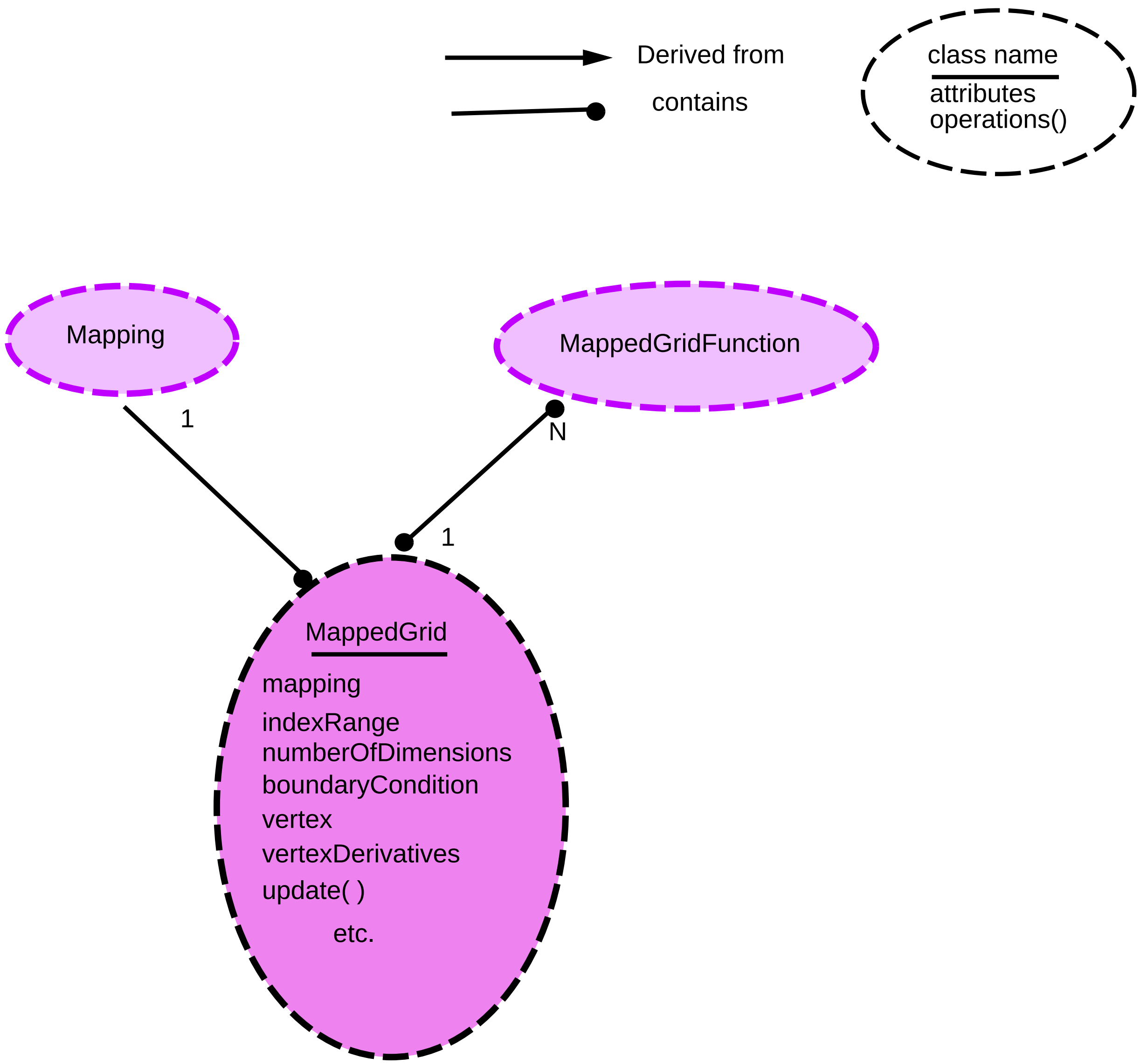


Figure 5: Class diagram for a MappedGrid

3.2 Example 2: grid functions

In the next example we introduce the notion of a grid function. A "realCompositeGridFunction" is a discrete function that lives on the grid points (or cell centres or faces) of a CompositeGrid [5]. A `realCompositeGridFunction` contains a list of "realMappedGridFunctions", one `realMappedGridFunction` for each component grid.

(file `Overture/primer/example2.C`)

```
#include "Overture.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------ \n");
  printf(" Read an overlapping grid from a data base file               \n");
  printf(" Make a grid function, assign it values and then display it   \n");
  printf(" ------------------------------------------------------------ \n");

  aString nameOfOGFile;
  cout << "Enter the name of the overlapping grid data base file " << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  realCompositeGridFunction u(cg);                      // create a composite grid function
  u=0.;                                                 // initialize to zero
  Index I1,I2,I3;                                       // A++ Index object

  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )  // loop over component grids
  {
    getIndex(cg[grid].indexRange(),I1,I2,I3);                  // assign I1,I2,I3 from indexRange
    u[grid](I1,I2,I3)=sin(cg[grid].vertex()(I1,I2,I3,axis1))   // assign all interior points on this
                     *cos(cg[grid].vertex()(I1,I2,I3,axis2));  // component grid
  }
  u.display("here is u=sin(x)*cos(y)");

  Overture::finish();
  return 0;
}
```

A "realCompositeGridFunction will either be a grid function of "floats" or a grid function of "doubles" depending on a compiler flag. In the above example we use the function `getIndex` to define the Index objects `I1,I2,I3` corresponding to the `indexRange` – i.e. the interior points of the grid. The interior points on each component grid of the grid function `u` are given values equal to $sin(x)cos(y)$. The object `u[grid]` is a "realMappedGridFunction". This object is derived from an A++ array and thus inherits all the functionality of an A++ array.

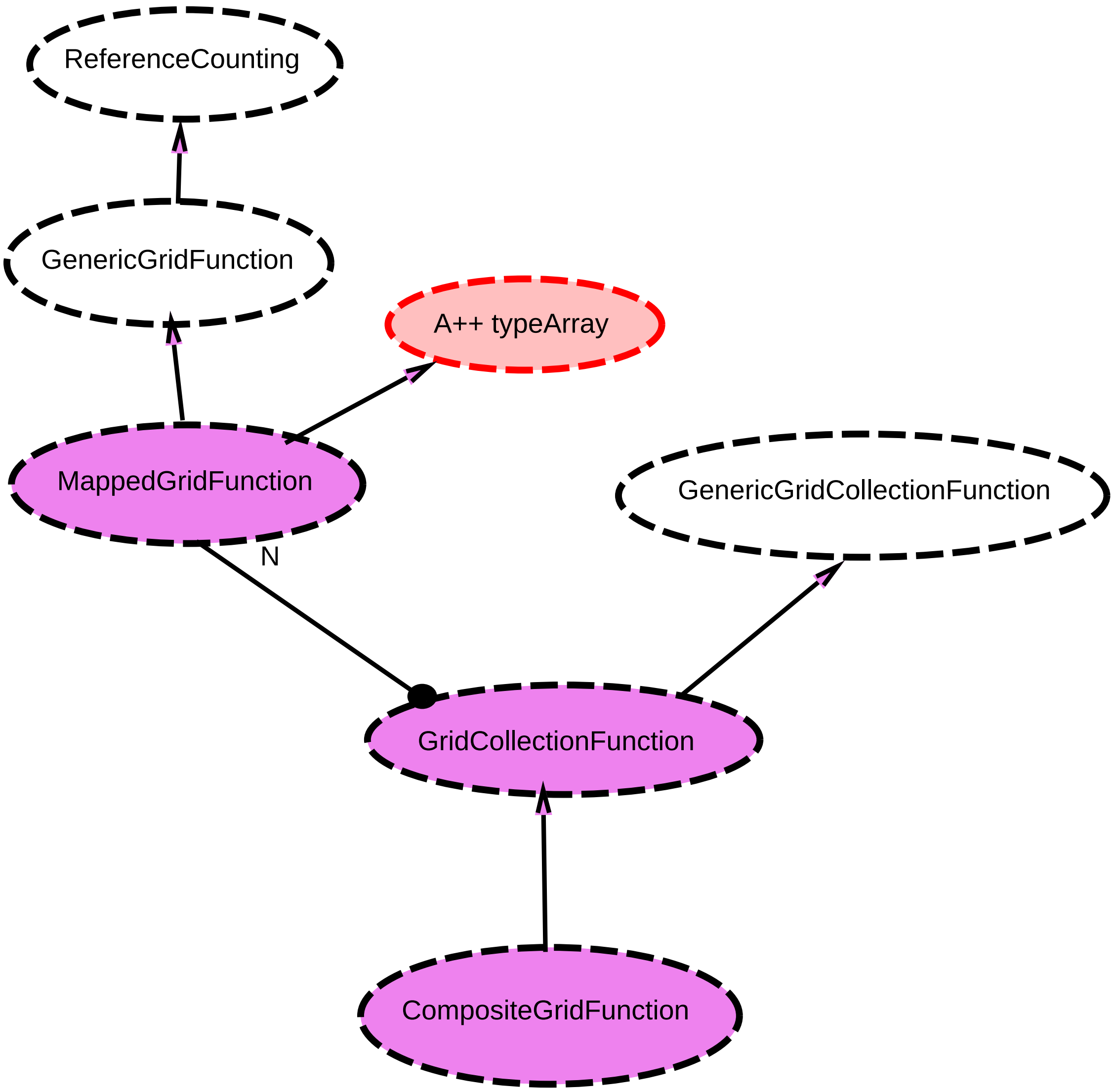


Figure 6: Class diagram for grid function classes

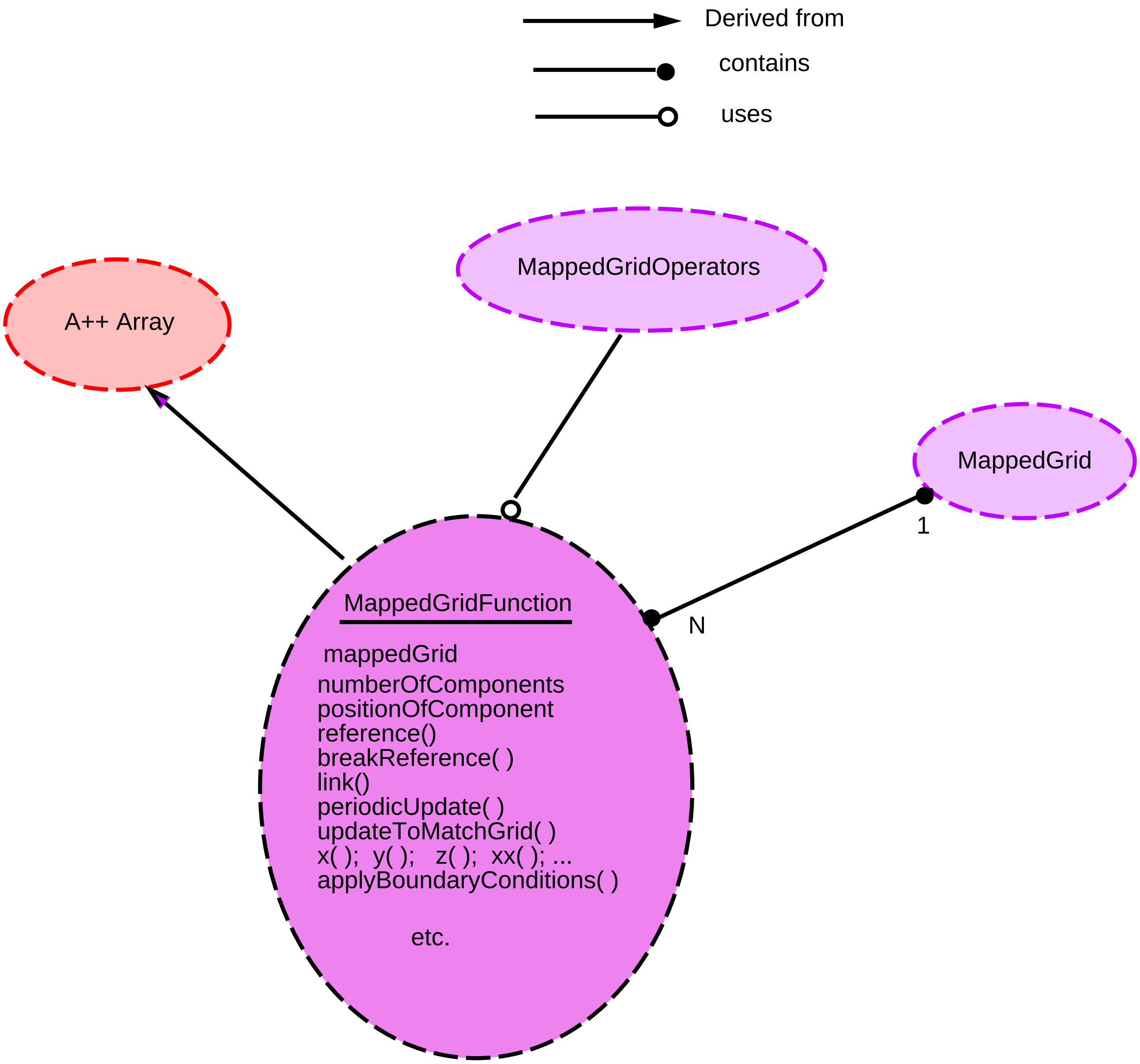


Figure 7: Class diagram for a MappedGridFunction

3.3 Example 3: interpolation

In the next example we show how to interpolate a grid function, i.e. how to obtain the values at the interpolation points given the values at all other points. In order to interpolate you must first create an “Interpolant” object. This object knows how to interpolate grid functions on a given CompositeGrid.

(file `Overture/primer/example3.C`)

```
#include "Overture.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------ \n");
  printf(" Demonstrate how to interpolate a compositeGridFunction       \n");
  printf(" ------------------------------------------------------------ \n");

  aString nameOfOGFile;
  cout << "Enter the name of the overlapping grid data base file " << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  realCompositeGridFunction u(cg);        // create a composite grid function
  u=0.;                                   // initialize to zero
  Index I1,I2,I3;                         // A++ Index object

  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )  // loop over component grids
  {
    getIndex(cg[grid].indexRange(),I1,I2,I3);                    // assign I1,I2,I3
    where( cg[grid].mask()(I1,I2,I3) > 0 )                       // only assign points with mask>0
      u[grid](I1,I2,I3)=sin(cg[grid].vertex()(I1,I2,I3,axis1))   // do not assign interpolation points
                       *cos(cg[grid].vertex()(I1,I2,I3,axis2));
  }
  u.display("here is u=sin(x)*cos(y) before interpolation");

  // Interpolant interpolant(cg);      // Make an interpolant
  Interpolant & interpolant = *new Interpolant(cg);               // do this instead for now.
  interpolant.interpolate(u);       // interpolate
  u.display("here is u after interpolation");

  u.interpolate();     // another way to interpolate, same result as above
  u.display("here is u after interpolate, version 2");

  Overture::finish();
  return 0;
}
```

In this example we use the `mask` array to selectively assign the grid points. The mask array is positive for discretization points, negative for interpolation points and zero for unused points. After interpolation the values at points with `mask<0` will have been assigned. As shown in the example there are two ways to interpolate. The second way, `u.interpolate()` may seem a bit mysterious since why should the grid function know about the Interpolant? The answer is that when the Interpolant is made it tells the CompositeGrid that it exists. The grid function checks with the CompositeGrid that it is associated with to see if an Interpolant has been made and if so it uses it.

3.4 Example 4: show files

Grid functions can be saved in a “show file” and later displayed with plotStuff (in the Overture/bin directory). In this example we show how to make a show file. More work has to be done on show files so some of the syntax may change in the future.

(file `Overture/primer/example4.C`)

```
#include "Overture.h"
#include "Ogshow.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ----------------------------------------------------------------- \n");
  printf(" Demonstrate how to use the Ogshow class to save results in a file \n");
  printf(" to be later plotted with plotSuff                                 \n");
  printf(" ----------------------------------------------------------------- \n");

  aString nameOfOGFile, nameOfShowFile;
  cout << "example4>> Enter the name of the (old) overlapping grid file:" << endl;
  cin >> nameOfOGFile;
  cout << "example4>> Enter the name of the (new) show file (blank for none):" << endl;
  cin >> nameOfShowFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  Ogshow show( nameOfShowFile );  // create a show file

  show.saveGeneralComment("Solution to the Navier-Stokes"); // save a general comment in the show file
  show.saveGeneralComment(" file written on April 1");      // save another general comment

  Range all;                                 // a null Range is used to dimension the grid function
  const int numberOfComponents=3;
  realCompositeGridFunction q(cg,all,all,all,numberOfComponents); // create a grid function with 3 components
  q=0.;

  realCompositeGridFunction u,v,machNumber;  // create grid functions for components
  u.link(q,Range(0,0));                      // link u to the first component of q
  v.link(q,Range(1,1));                      // link v to the second component of q
  machNumber.link(q,Range(2,2));             // ...
  q.setName("q");                            // assign name to grid function and components
  q.setName("u",0);                          // name of first component
  q.setName("v",1);                          // name of second component
  q.setName("Mach Number",2);                // name of third component

  char buffer[80];                           // buffer for sPrintF
  Index I1,I2,I3;
  int numberOfTimeSteps=5;
  for( int i=1; i<=numberOfTimeSteps; i++ )  // Now save the grid functions at different time steps
  {
    show.startFrame();                       // start a new frame
    real t=i*.1;
    show.saveComment(0,sPrintF(buffer,"Here is solution %i",i));   // comment 0 (shown on plot)
    show.saveComment(1,sPrintF(buffer,"  t=%e ",t));               // comment 1 (shown on plot)
    for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )      // loop over component grids
    {
      getIndex(cg[grid].indexRange(),I1,I2,I3);
      u[grid](I1,I2,I3)=sin(twoPi*(cg[grid].vertex()(I1,I2,I3,axis1)-t))   // assign u on each grid
                       *cos(twoPi*(cg[grid].vertex()(I1,I2,I3,axis2)+t));
    }
    v=u*2.;
    machNumber=u*u+v*v;
    show.saveSolution( q );              // save the current grid function
  }

  Overture::finish();
  return 0;
```

```

}
```

This example demonstrates a few other features of grid functions such as declaring a grid function with more than one component and linking one grid function to another.

Use plotStuff to display the results from this example, “plotStuff fileName.show” where “fileName.show” was the name chosen for the showfile when the program was run.

3.5 Example 5: Differentiating grid functions

The `MappedGridOperators` and `CompositeGridOperators` classes can be used can be used to compute spatial derivatives of grid functions and to apply boundary conditions. In this example we show how to differentiate grid functions to second or fourth-order accuracy.

(file `Overture/primer/example5.C`)

```
#include "Overture.h"
#include "CompositeGridOperators.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" -------------------------------------------------------------------------- \n");
  printf(" Demonstrate the operators for taking derivatives of compositeGridFunction's \n");
  printf(" -------------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "example5>> Enter the name of the (old) overlapping grid file:" << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  CompositeGridOperators operators(cg);                        // operators for a CompositeGridFunction
  Range all;
  realCompositeGridFunction u(cg),ux(cg),w(cg,all,all,all,2);  // create some composite grid functions

  u.setOperators(operators);                                   // tell grid function which operators to use
  w.setOperators(operators);

  u=1.;
  ux=u.x();                                                    // compute the x derivative of u
  ux.display("Here is the x derivative of u=1 (computed at interior and boundary points)");
  w=2.;
  w.y().display("Here is the y derivative of w");
  Range c0(0,0),c1(1,1);
  w.y(c0).display("Here is the y derivative of component 0 of w");
  w.y(c1).display("Here is the y derivative of component 1 of w");

  real error;
  Index I1,I2,I3;
  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )        // loop over component grids
  {
    MappedGrid & mg = cg[grid];
    getIndex(mg.dimension(),I1,I2,I3);                           // assign I1,I2,I3 for dimension
    u[grid](I1,I2,I3)=sin(mg.vertex()(I1,I2,I3,axis1))*cos(mg.vertex()(I1,I2,I3,axis2));
    getIndex(mg.indexRange(),I1,I2,I3);                          // assign I1,I2,I3 for indexRange
    operators.setOrderOfAccuracy(2);                           // set order of accuracy to 4

    ux[grid](I1,I2,I3)=u[grid].x()(I1,I2,I3);                  // here is the x derivative of u[grid]

    error = max(fabs( ux[grid](I1,I2,I3)- cos(mg.vertex()(I1,I2,I3,axis1))*cos(mg.vertex()(I1,I2,I3,axis2) )));
    cout << "Maximum error (2nd order) = " << error << endl;

    error = max(fabs( operators[grid].x(u[grid])(I1,I2,I3)      // another way to compute derivatives
                - cos(mg.vertex()(I1,I2,I3,axis1))*cos(mg.vertex()(I1,I2,I3,axis2) )));
    cout << "Maximum error (2nd order) = " << error << endl;

    operators.setOrderOfAccuracy(4);                           // set order of accuracy to 4
    getIndex(mg.indexRange(),I1,I2,I3,-1);                       // decrease ranges by 1 for 4th order
    error = max(fabs(u[grid].x()(I1,I2,I3)-cos(mg.vertex()(I1,I2,I3,axis1))*cos(mg.vertex()(I1,I2,I3,axis2))));
    cout << "Maximum error (4th order) = " << error << endl;
  }
  Overture::finish();
  return 0;
}
```

In this example we create a `CompositeGridOperators` object and associate it with a `CompositeGrid`. We compute the x-derivative of a `realCompositeGridFunction` and of `realMappedGridFunction`'s. The member function "x" in the grid function returns the x derivative of the grid function as a new grid function. It uses the derivative defined in the `CompositeGridOperators` object which in turn uses a `MappedGridOperators` object to compute the derivatives of a MappedGridFunction. The default `MappedGridOperators` object used by a `CompositeGridOperators` can be changed. Note that by default the derivative of a `realCompositeGridFunction` is only computed at interior and boundary points (indexRange). Thus to access (make a view) of the derivative values of the grid function `u.x()` at the Index's `(I1,I2,I3)` it is necessary to say `u.x()(I1,I2,I3)`. On the other hand the statement `u.x(I1,I2,I3)` will evaluate the derivatives on the points defined by `(I1,I2,I3)`, but will return a grid function that is dimensioned for the entire grid. Thus in general on could say `u.x(I1,I2,I3)(J1,J2,J3)` to evaluate the derivatives at points `(I1,I2,I3)` but to use (take a view) of the grid function at the Index's `(J1,J2,J3)`.

The `MappedGridOperators` and `CompositeGridOperators` classes are described in more detail in the grid function documentation.

3.6 Example 6: Solving a simple PDE using Differential and Boundary operators

In this example we solve the convection-diffusion equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} = \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$$

with a simple time stepping method (forward Euler) The solutions at different time steps are saved in a show file. (file `Overture/primer/example6.C`)

```
#include "Overture.h"
#include "Ogshow.h"
#include "CompositeGridOperators.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" -------------------------------------------------------------------------- \n");
  printf("Solve: u.t + a*u.x + b*u.y = viscosity*( u.xx + u.yy ) on an Overlapping grid \n");
  printf("Save results in a show file, use plotStuff to view this file                \n");
  printf(" -------------------------------------------------------------------------- \n");

  aString nameOfOGFile="cice.hdf", nameOfShowFile="example6.show";
  #ifndef USE_PPP
   // prompt for name changes in serial, for parallel just use default
   cout << "example6>> Enter the name of the (old) overlapping grid file:" << endl;
   cin >> nameOfOGFile;
   cout << "example6>> Enter the name of the (new) show file (blank for none):" << endl;
   cin >> nameOfShowFile;
  #endif

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  // Interpolant interpolant(cg);                                 // Make an interpolant
  Interpolant & interpolant = *new Interpolant(cg);               // do this instead for now.

  Ogshow show( nameOfShowFile );                               // create a show file
  show.saveGeneralComment("Convection Diffusion Equation");    // save a general comment in the show file
//  show.setFlushFrequency(10);                                   // flush file every 10 frames

  CompositeGridOperators operators(cg);                        // operators for a CompositeGrid
  // operators.setOrderOfAccuracy(4);                          // for fourth order

  Range all;
  realCompositeGridFunction u(cg,all,all,all,1);               // create a grid function
  u.setOperators(operators);
  u.setName("u");                                              // name the grid function

  u=1.;                                                        // initial condition
  real t=0, dt=.001;                                           // initialize time and time step
  real a=1., b=1., viscosity=.1;                               // initialize parameters

  char buffer[80];                                             // buffer for sprintf
  int numberOfTimeSteps=200;
  for( int i=0; i<numberOfTimeSteps; i++ )                    // take some time steps
  {
    if( i % 40 == 0 )  // save solution every 40 steps
    {
      show.startFrame();                                         // start a new frame
      show.saveComment(0,sPrintF(buffer,"Here is solution %i",i));   // comment 0 (shown on plot)
      show.saveComment(1,sPrintF(buffer,"  t=%e ",t));               // comment 1 (shown on plot)
      show.saveSolution( u );                                        // save the current grid function
    }
    u+=dt*( -a*u.x() - b*u.y() + viscosity*(u.xx() + u.yy())); // take a time step with Euler's method
    t+=dt;
    u.interpolate();                                           // interpolate
    // apply a dirichlet BC on all boundaries:
```

```
      u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
      // u.applyBoundaryCondition(0,BCTypes::extrapolate,BCTypes::allBoundaries,0.); // for 4th order
      u.finishBoundaryConditions();
    }

    Overture::finish();
    return 0;

  }
```

To run this example:

- First create an overlapping grid. For example, from the `Overture/sampleGrids` directory type `../bin/ogen noplot cic.cmd` to create the grid `cic.hdf` using the command file `cic.cmd`.
- Run `example6`. Enter `cic.hdf` as the overlapping grid to use and `primer.show` as the name of the output "show" file.
- Look at the results by typing `../bin/plotStuff primer.show`.

Note that a fixed time step is used in this example and that the time step may not be small enough to keep the method stable if you use a different grid from `cic.hdf`.

Currently, computing derivatives in this way will not be so efficient. An efficient way to compute derivatives is described in section (**??**) and in the grid function documentation.

In this example we chose the boundary conditions to be dirichlet on all sides of all grids. By default the values at dirichet boundaries are set to zero. Boundary conditions can be defined in a much more general manner as described in the grid function documentation.

If you want to solve the problem with fourth-order accuracy you can un-comment the two lines indicated in example6.C. You will need to use an overlapping grid created for fourth order (such as `cic.4.cmd`) and you will need to decrease the time step `dt` by a factor of 4 or so(?).

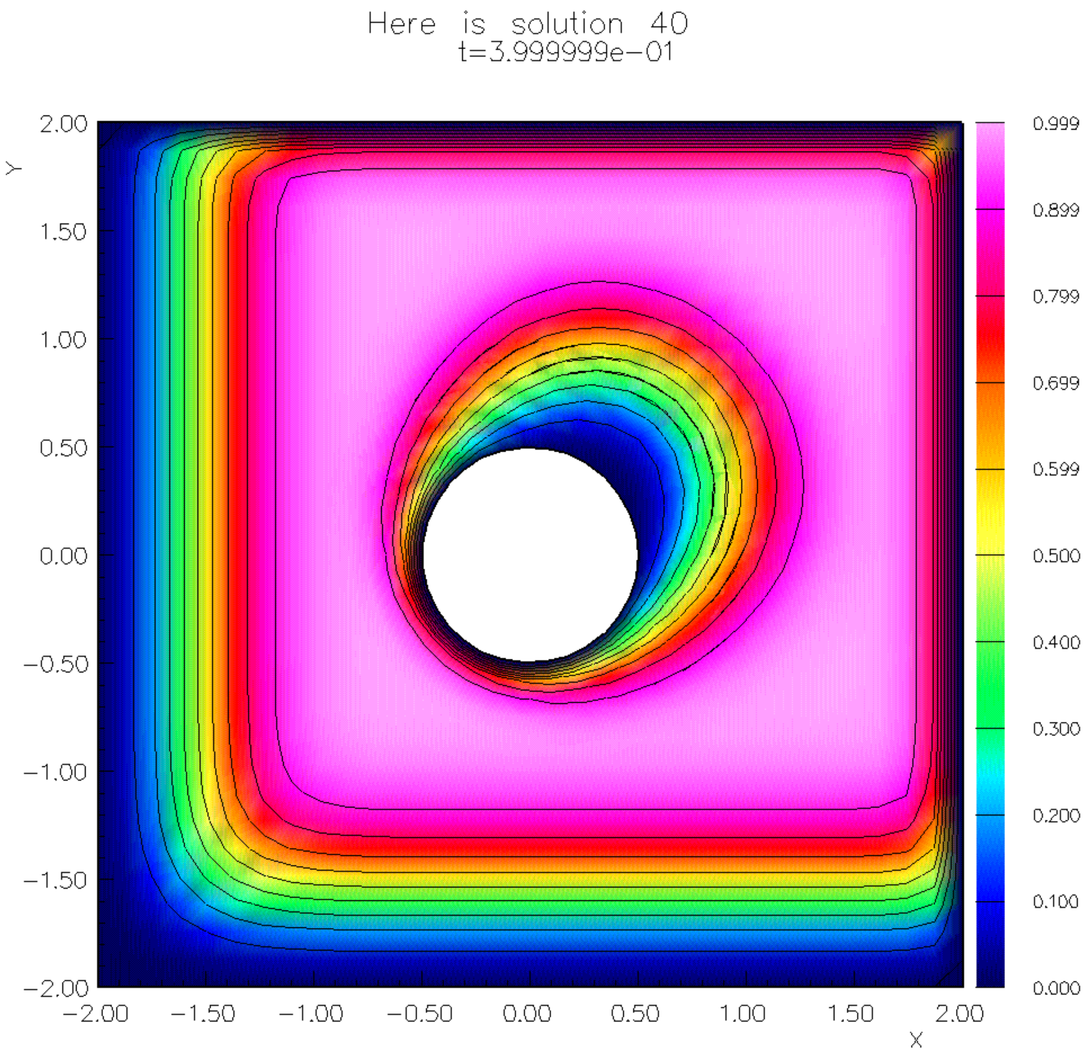


Figure 8: Results from example6. Solving a convection diffusion equation.

3.7 Example 7: Solving Poisson's equation with Oges

In this example we solve Poisson's equation in 2 or 3D,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f \quad \text{for } \mathbf{x} \in \Omega$$

with Dirichlet boundary conditions

$$u = 0 \quad \text{for } \mathbf{x} \in \partial\Omega$$

(file `Overture/primer/example7.C`)

```
#include "Overture.h"
#include "CompositeGridOperators.h"
#include "Oges.h"
#include "display.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ----------------------------------------------------------------------------- \n");
  printf("Use the operators to create the matrix for the discrete Laplacian operator with\n");
  printf("  boundary conditions.                                                         \n");
  printf("Use the Oges class to solve the system of equations.                           \n");
  printf(" ----------------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "example7>> Enter the name of the (old) overlapping grid file:" << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  // make a grid function to hold the coefficients
  Range all;
  int stencilSize=int( pow(3,cg.numberOfDimensions())+1.5 );  // add 1 for interpolation equations
  realCompositeGridFunction coeff(cg,stencilSize,all,all,all);
  coeff.setIsACoefficientMatrix(TRUE,stencilSize);

  // create grid functions:
  realCompositeGridFunction u(cg),f(cg);

  CompositeGridOperators op(cg);                            // create some differential operators
  op.setStencilSize(stencilSize);
  coeff.setOperators(op);

  coeff=op.laplacianCoefficients();       // get the coefficients for the Laplace operator
  // make some shorter names for readability
  BCTypes::BCNames dirichlet             = BCTypes::dirichlet,
                   extrapolate           = BCTypes::extrapolate,
                   allBoundaries         = BCTypes::allBoundaries;

  // fill in the coefficients for the boundary conditions
  coeff.applyBoundaryConditionCoefficients(0,0,dirichlet,  allBoundaries);
  coeff.applyBoundaryConditionCoefficients(0,0,extrapolate,allBoundaries); // extrap ghost line
  coeff.finishBoundaryConditions();

  Oges solver( cg );                     // create a solver
  // solver.set(OgesParameters::THEkeepSparseMatrix,true);  // turn this on if we want to save the matrix

  solver.setCoefficientArray( coeff );   // supply coefficients


  // assign the rhs:  Laplacian(u)=1, u=0 on the boundary
  Index I1,I2,I3;
  Index Ib1,Ib2,Ib3;
  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )
  {
```

```
    // displayCoeff(coeff[grid],sPrintF("coeff for grid=%i",grid),stdout); // un-comment this line to display the equations

    MappedGrid & mg = cg[grid];
    getIndex(mg.indexRange(),I1,I2,I3);

    f[grid](I1,I2,I3)=1.;
    for( int side=Start; side<=End; side++ )
    for( int axis=axis1; axis<cg.numberOfDimensions(); axis++ )
    {
      if( mg.boundaryCondition()(side,axis) > 0 )
      {
        getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
        f[grid](Ib1,Ib2,Ib3)=0.;
      }
    }
  }

  solver.solve( u,f );   // solve the equations

  // solver.writeMatrixToFile("matrix.dat");  // save the sparse matrix to a file (uncomment the line above too)

  //

  u.display("Here is the solution to Laplacian(u)=1, u=0 on the boundary");

  Overture::finish();
  return(0);

}
```

We use the `Oges` (Overlapping grid equation solver) class to use a sparse matrix solver to solve the problem. We use the differential operators in the `CompositeGridOperators` class to define coefficients of the Laplacian operator and the coefficients for the boundary condition. By default the `Oges` solver will use the Yale sparse matrix solver. The first time the problem is solved the matrix will be factored. Subsequent calls to solve with different right-hand-sides will only involve a back-substitution. See the `Oges` documentation for further details on the many available options.

If instead of the Laplacian operator we wanted to define some other operator, say,

$$2\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+3\frac{\partial u}{\partial x}$$

then we could have used the statement

```
coeff=2.*u.xxCoefficients()+u.yyCoefficients()+3.*u.xCoefficients(();
```

3.8 Example 8: Interactive plotting with PlotStuff

In this example we show how to plot "stuff" interactively from a program using the `PlotStuff` class. These plotting routines are based on OpenGL and can run on many platforms. Currently on Sun's I use Brian Paul's Mesa library which is a public domain implementation of OpenGL that runs under X-windows. More information about plotting can be found in the document /home/henshaw.0/Overture/ogshow/PlotStuff.tex.

Here is an example code that uses the `PlotStuff` class to plot various objects from the Overture class (file `Overture/primer/example8.C`)

```
#include "Overture.h"
#include "PlotStuff.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------------- \n");
  printf("Demonstrate interactive plotting using the PlotStuff class           \n");
  printf("Make a menu and selectively plot the grid or contours or streamlines.\n");
  printf(" ------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "example>> Enter the name of the (old) overlapping grid file:" << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  Range all;
  realCompositeGridFunction u(cg,all,all,all,2);           // create a grid function with 2 components
  u.setName("Velocity Stuff");                             // give names to grid function ...
  u.setName("u Stuff",0);                                  // ...and components
  u.setName("v Stuff",1);
  Index I1,I2,I3;
  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )  // loop over component grids
  {
    getIndex(cg[grid].dimension(),I1,I2,I3);                        // assign I1,I2,I3 from dimension
    u[grid](I1,I2,I3,0)=sin(Pi*cg[grid].center()(I1,I2,I3,axis1))   // component 0 : sin(x)*cos(y)
                       *cos(Pi*cg[grid].center()(I1,I2,I3,axis2));
    u[grid](I1,I2,I3,1)=cos(Pi*cg[grid].center()(I1,I2,I3,axis1))   // component 1 : cos(x)*sin(y)
                       *sin(Pi*cg[grid].center()(I1,I2,I3,axis2));
  }

  bool openGraphicsWindow=TRUE;
  PlotStuff ps(openGraphicsWindow,"example8");  // create a PlotStuff object
  PlotStuffParameters psp;                      // This object is used to change plotting parameters

  aString answer;
  aString menu[] = {
                    "!example8",
                    "contour",                  // Make some menu items
                    "stream lines",
                    "grid",
                    "read command file",
                    "save command file",
                    "erase",
                    "exit",
                    "" };                       // empty string denotes the end of the menu
  for(;;)
  {
    ps.getMenuItem(menu,answer);                // put up a menu and wait for a response
    if( answer=="contour" )
    {
      psp.set(GI_TOP_LABEL,"My Contour Plot");  // set title
      PlotIt::contour(ps,u,psp);                        // contour/surface plots
    }
    else if( answer=="grid" )
    {
```

```
      PlotIt::plot(ps,cg);                              // plot the composite grid
    }
    else if( answer=="stream lines" )
    {
      PlotIt::streamLines(ps,u);                        // streamlines
    }
    else if( answer=="read command file" )
    {
      ps.readCommandFile();
    }
    else if( answer=="save command file" )
    {
      ps.saveCommandFile();
    }
    else if( answer=="erase" )
    {
      ps.erase();
    }
    else if( answer=="exit" )
    {
      break;
    }
  }

  Overture::finish();
  return 0;
}

```

When the example is run a window will pop up. To see the menus, put the cursor over the window and press the right mouse button. Choosing **contour**, for example, will cause the `contour` function to be called. Now choose **plot** to display the contour/surface plot. Other menu items allow one to change features of the plot. Buttons on the window allow one to shift rotate and zoom the plot. The left mouse button can be used to zoom using a rubber-band box.

3.9 Example 9: Saving and Reading a Restart file

This example show how to save information in a data base file. This could be a restart file for a PDE solver. It is also possible to create a hierarchical directory structure within the data base file. See the data base documentation for further details.

See also the documentation on ShowFileReader in the Ogshow documentation for how to read grids and grid functions from a show file. This would be another way to get initial conditions for a solver.

(file `Overture/primer/example9.C`)

```
#include "Overture.h"
#include "HDF_DataBase.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ---------------------------------------------------------------------- \n");
  printf("This example shows how to save information in a data base file.          \n");
  printf("This could be a restart file for a PDE solver                            \n");
  printf("See also the documentation on ShowFileReader in the Ogshow documentation \n");
  printf("for how to read grids and grid functions from a show file. This would    \n");
  printf("be another way to get initial conditions for a solver.                   \n");
  printf(" ---------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "example>> Enter the name of the (old) overlapping grid file:" << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  Range all;
  realCompositeGridFunction u(cg,all,all,all,2);          // create a grid function with 2 components

//  u.setName("Velocity Stuff");                             // give names to grid function ...
//  u.setName("u Stuff",0);                                  // ...and components
//  u.setName("v Stuff",1);
  Index I1,I2,I3;
  for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )  // loop over component grids
  {
    getIndex(cg[grid].dimension(),I1,I2,I3);                   // assign I1,I2,I3 from dimension
    const realArray & x =cg[grid].center()(I1,I2,I3,axis1);
    const realArray & y =cg[grid].center()(I1,I2,I3,axis2);

    u[grid](I1,I2,I3,0)=sin(Pi*x)*cos(Pi*y);                  // component 0
    u[grid](I1,I2,I3,1)=cos(Pi*x)*sin(Pi*y);                  // component 1
  }

  // Here are some other things that we want to save:
  realArray v(20,20);         // distributed array (parallel)
  v=1.;

  realSerialArray drag(100);  // serial array
  drag=3.;

  real time=55.6;
  aString comment = "my restart file";

  HDF_DataBase db;   // make a data base

  printF("Mount file example9.hdf and save some data...\n");

  db.mount("example9.hdf","I");    // open the data base, I=initialize

  cg.put(db,"My Grid");            // save a grid
  u.put(db,"My Solution");         // save a grid function
  db.putDistributed(v,"v");        // save a distributed array of data
  db.put(drag,"drag");             // save an array of data
```

```
  db.put(time,"time");             // save a real number
  db.put(comment,"comment");       // save a string

  cout << "Close the file example9.hdf...\n";
  db.unmount();                    // close the data base


// Now mount the file and read back the data

  HDF_DataBase db2;
  cout << "Mount file example9.hdf and read back some data...\n";
  db2.mount("example9.hdf","R");   // mount, R=read-only

  // define new objects to read the data into
  CompositeGrid cg2;
  realCompositeGridFunction u2;

  realSerialArray drag2;
  realArray v2;
  real time2;
  aString comment2;

  // note that the data can be read back in any order
  db2.getDistributed(v2,"v");            // save an array of data
  db2.get(drag2,"drag");            // save an array of data
  db2.get(time2,"time");            // save a real number
  db2.get(comment2,"comment");      // save a string

  cg2.get(db2,"My Grid");

  u2.updateToMatchGrid(cg2);       // *** note: do an update before reading in the grid
  u2.get(db2,"My Solution");
  u2.display("Here is u2");

  db2.unmount();                    // close the file

  // now check that we have read back the data properly

  if( max(fabs(u2-u))==0. && max(abs(drag2-drag))==0. && fabs(time2-time)==0. && comment==comment2 )
    printF("Objects were successfully read back in.\n");
  else
    printF("ERROR: Objects were NOT successfully read back in.\n");

  Overture::finish();
  return 0;
}
```

3.10 Example: 2D Wave equation (optimised for performance and memory usage)

This example shows how to solve the 2D wave equation,

$$u_{tt} - c^2 \Delta u = 0 \ .$$

This example has been optimised for performance and memory usage. Rectangular grids especially, are treated in an efficent manner.

We discretize this second order equation in time using a second-order centered difference,

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} - c^2 \Delta_h u_i^n = 0 \ ,$$

where the superscript n denotes the time level. We use either a 2nd order or 4th order discretization in space. We only interpolate assuming a second-order scheme so formally the method is not fourth-order.

The above scheme has no dissipation. To smooth out numerical oscillations an artificial dissipation term has been added of the form $h^4 \partial_t u_{xxxx}$,

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} - c^2 \Delta_h u_i^n + -C \ h^4 (D_+ D_-)^2 (u^n - u^{n-1}) = 0$$

To allow the use of a fourth-order difference approximation on an overlapping grid that is only built for a stencil that is 3×3 we use the `extrapolateInterpolationNeighbours` boundary condition to extrapolate values at the normally unused points next to interpolation points. With these values defined we can apply a fourth order difference at all interior points.

See file `Overture/primer/wave.C`.

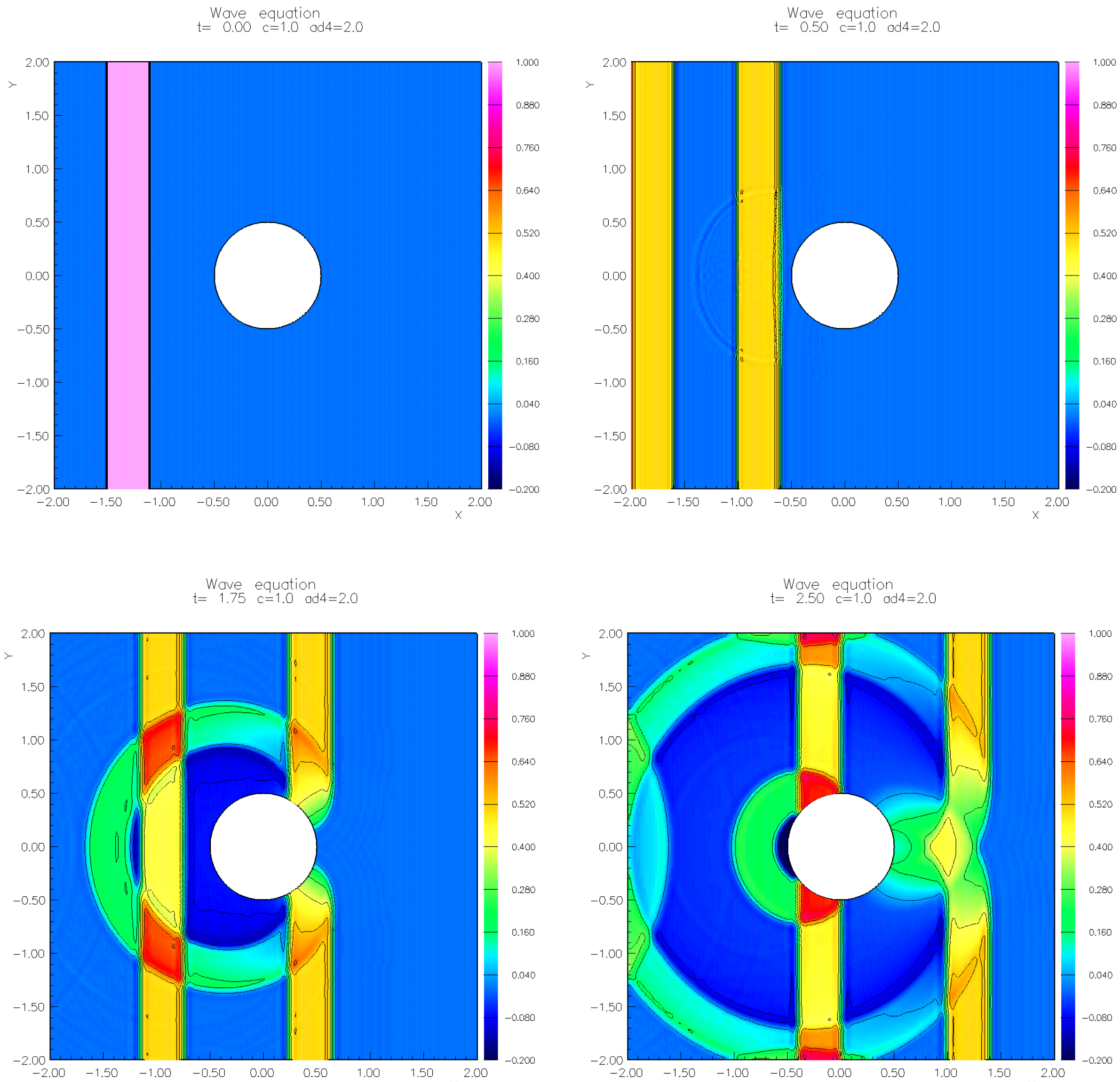


Figure 9: Results from the 2D wave equation. An initial rectangular pulse splits into two and then collides with the cylinder and walls.

3.11 Example: Moving overlapping grids

This example shows how to move a component grid and recompute the overlapping grid. The second component grid will be rotated.

You might try running this example with the grid `sis.hdf` which is an overlapping grid for a square inside a square. When the example runs a window will pop up and a grid will be shown. Choose the menu item `erase and exit` (right mouse button) to continue.

The results will also be saved in a show file, called “move1.show”. Use Overture/bin/plotStuff to look at this file. (file `Overture/primer/move1.C`)

```
#include "Ogen.h"
#include "PlotStuff.h"
#include "MatrixTransform.h"
#include "Ogshow.h"
#include "ParallelUtility.h"

// =================================================================================================
// Moving Grid Example:
//   o read in a grid from a data-base file, rotate a component grid and recompute the overlapping grid.
//   o interpolate a grid function, update the interpolate for the new grid
//   o save solutions in a show file
//
// Usage:
//   move1 [-grid=<name>] [-numSteps=<>] [-shift] [-rotate] [-interpolate=[0|1]] [-saveShow=[0|1]]
//
// Examples:
//   move1 -grid=cic
//   move1 -grid=sib
//   move1 -grid=ellipsoid1
//
//  mpirun -np 1 move1 -grid=cic -numSteps=5
//  mpirun -np 1 move1 -grid=cic -numSteps=5 -interpolate=1
//  srun -N1 -n2 -ppdebug move1 -grid=cic -numSteps=5
//  srun -N1 -n4 -ppdebug move1 -grid=cice -numSteps=5 -interpolate=1 -saveShow=0
//  totalview srun -a -N1 -n1 -ppdebug move1 -grid=cic
// =================================================================================================
int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  Mapping::debug=0;
  int debug=0;

  int numGhost=2;  // for second-order accurate (1 is good enough for implicit)
  MappedGrid::setMinimumNumberOfDistributedGhostLines(numGhost);

  printF(" moving grid demo: \n"
         " move1 [-grid=<name>] [-numSteps=<>] [-shift] [-rotate] [-interpolate=[0|1]] [-saveShow=[0|1]]\n");

  enum
  {
    rotate,
    shift
  } moveOption=shift;

  int numberOfSteps=20;
  real deltaAngle=5.*Pi/180.;
  real deltaShift=-.01;
  int useFullAlgorithmInterval=10; // 10000;
  #ifdef USE_PPP
    useFullAlgorithmInterval=1;  // for now always use full algorithm for ogen
  #endif

  // aString nameOfOGFile = "cice2.order2.hdf";
  aString nameOfOGFile = "cic.hdf";

  int plotOption=true;
  int interpolate=false;
  int saveShow=true;
  if( argc > 1 )
```

```
{ // look at arguments for "noplot"
  aString line;
  int len=0;
  for( int i=1; i<argc; i++ )
  {
    line=argv[i];
    if( line=="-noplot" || line=="noplot" )
      plotOption=false;
    else if( len=line.matches("-grid=") )
    {
      nameOfOGFile=line(len,line.length()-1);
    }
    else if( line=="-shift" )
    {
      moveOption=shift;
    }
    else if( line=="-rotate" )
    {
      moveOption=rotate;
    }
    else if( len=line.matches("-numSteps=") )
    {
      sScanF(line(len,line.length()-1),"%i",&numberOfSteps);
    }
    else if( len=line.matches("-saveShow=") )
    {
      sScanF(line(len,line.length()-1),"%i",&saveShow);
    }
    else if( len=line.matches("-interpolate=") )
    {
      sScanF(line(len,line.length()-1),"%i",&interpolate);
    }
    else
    {
      printF("Unknown option=[%s]\n",(const char*)line);
    }


  }
}


PlotStuff ps(plotOption,"Moving Grid Example");         // for plotting
PlotStuffParameters psp;

// Create two CompositeGrid objects, cg[0] and cg[1]
CompositeGrid cg[2];
getFromADataBase(cg[0],nameOfOGFile);             // read cg[0] from a data-base file
cg[1]=cg[0];                                      // copy cg[0] into cg[1]

if( cg[0].numberOfDimensions()==2 )
  psp.set(GI_PLOT_INTERPOLATION_POINTS,true);

if( debug & 2  )
{
  psp.set(GI_TOP_LABEL,"initial grid");  // set title
  PlotIt::plot(ps,cg[0],psp);
}


// Move some component grids (do this by changing the mapping)
int gridsToMove[5]={1,2,3,4,5};  // move at most 5 grids
MatrixTransform *transform0[5]={NULL,NULL,NULL,NULL,NULL}; //
MatrixTransform *transform1[5]={NULL,NULL,NULL,NULL,NULL};

// By default we move all the grids but grid=0
int numberOfGridsToMove=cg[0].numberOfComponentGrids()-1;  // number of grids to move
assert( numberOfGridsToMove<=5 );

for( int g=0; g<numberOfGridsToMove; g++ )
{
  int grid=gridsToMove[g];
```

```
    // Use this MatrixTransform to change the existing Mapping, the MatrixTransform
    // can rotate/scale and shift any Mapping, keep a transform for each composite grid

    Mapping & mappingToMove = *(cg[0][grid].mapping().mapPointer);
    transform0[g] = new MatrixTransform(mappingToMove);
    transform1[g] = new MatrixTransform(mappingToMove);
    cg[0][grid].reference(*transform0[g]);
    cg[1][grid].reference(*transform1[g]);
  }
  cg[0].updateReferences();
  cg[1].updateReferences();


//   // Use this MatrixTransform to change the existing Mapping, the MatrixTransform
//   // can rotate/scale and shift any Mapping, keep a transform for each composite grid
//   MatrixTransform transform0(mappingToMove);
//   MatrixTransform transform1(mappingToMove);

//   // Replace the mapping of the component grid that we want to move:
//   cg[0][gridToMove].reference(transform0);
//   cg[0].updateReferences();
//   cg[1][gridToMove].reference(transform1);
//   cg[1].updateReferences();

  if( debug & 2 )
  {
    ps.erase();
    psp.set(GI_TOP_LABEL,"cg[0] after reference to transform");  // set title
    PlotIt::plot(ps,cg[0],psp);
  }


  // now we destroy all the data on the new grid -- it will be shared with the old grid
  // this is not necessary to do but it will save space
  cg[1].destroy(CompositeGrid::EVERYTHING);

  // we tell the grid generator which grids have changed
  LogicalArray hasMoved(cg[0].numberOfComponentGrids());
  hasMoved    = true;
  hasMoved(0) = false;

  char buff[80];

  // Here is the overlapping grid generator
  Ogen gridGenerator(ps);

  // update the initial grid, since the above reference destroys the mask
  gridGenerator.updateOverlap(cg[0]);

  if( debug & 2 )
  {
    ps.erase();
    psp.set(GI_TOP_LABEL,"cg[0] gridGenerator.updateOverlap");  // set title
    PlotIt::plot(ps,cg[0],psp);
  }


  // Here is an interpolant
  // Interpolant interpolant(cg[0]);
  Interpolant & interpolant = *new Interpolant(cg[0]);               // do this instead for now.
  realCompositeGridFunction u(cg[0]);
  // Here is a show file
  Ogshow show("move1.show");
  show.setIsMovingGridProblem(true);


  // ---- Move the grid a bunch of times.----
  real angle=0.; // total angle rotated so far
  real xShift=0;  // cummulative shift
  for (int i=1; i<=numberOfSteps; i++)
  {
    int newCG = i % 2;        // new grid
```

```
    int oldCG = (i+1) % 2;    // old grid

    printF("--- take step %i\n",i);

    if( plotOption && i>0 )
    {
      ps.erase();
      psp.set(GI_TOP_LABEL,sPrintF(buff,"Solution at step=%i",i));  // set title
      PlotIt::plot(ps,cg[oldCG],psp);      // plot the current overlapping grid
      psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,true);     // set this to run in "movie" mode (after first plot)
      ps.redraw(true);
    }

    // Move the grids by changing the Mapping (rotate/shift)
    for( int g=0; g<numberOfGridsToMove; g++ )
    {
      MatrixTransform & transform = newCG==0 ? *transform0[g] : *transform1[g];
      if( moveOption==rotate )
      {
        angle += deltaAngle;
        transform.reset();  // reset transform since otherwise rotate is incremental
        transform.rotate(axis3,angle);
      }
      else
      {
        xShift += deltaShift;
        // printF(" xShift=%9.3e\n",xShift);
        transform.reset();  // reset transform since otherwise shift is incremental
        transform.shift(xShift,0.,0.);
      }
    }

    // Update the overlapping newCG, starting with and sharing data with oldCG.
    Ogen::MovingGridOption option = Ogen::useOptimalAlgorithm;
    if( i% useFullAlgorithmInterval == useFullAlgorithmInterval-1  )
    {
      printF("\n +++++++++++ use full algorithm in updateOverlap +++++++++++++++ \n");
      option=Ogen::useFullAlgorithm;
    }
    gridGenerator.updateOverlap(cg[newCG], cg[oldCG], hasMoved, option );

    if( interpolate )
    {
      interpolant.updateToMatchGrid(cg[newCG]);
      u.updateToMatchGrid(cg[newCG]);
    }

    // assign values to u
    Index I1,I2,I3;
    for( int grid=0; grid<cg[newCG].numberOfComponentGrids(); grid++ )
    {
      MappedGrid & mg = cg[newCG][grid];
      getIndex(mg.dimension(),I1,I2,I3);
      realSerialArray vertexLocal; getLocalArrayWithGhostBoundaries(mg.vertex(),vertexLocal);
      realSerialArray uLocal; getLocalArrayWithGhostBoundaries(u[grid],uLocal);
      bool ok = ParallelUtility::getLocalArrayBounds(u[grid],uLocal,I1,I2,I3,1);
      if( ok ) // this processor has grid points
      {
        real freq = 2.*i/numberOfSteps;
        uLocal(I1,I2,I3)=cos(freq*vertexLocal(I1,I2,I3,axis1))*sin(freq*vertexLocal(I1,I2,I3,axis2));
      }

    }
    if( interpolate )
      u.interpolate();
    // save the result in a show file, every fourth step
    if( saveShow && (i % 4) == 1 )
    {
      show.startFrame();
      show.saveComment(0,sPrintF(buff,"Solution form move1 ar step = %i",i));
      show.saveSolution(u);
    }
```

```
  }
  if( saveShow )
    printF("Results saved in move1.show, use Overture/bin/plotStuff to view this file\n");
  show.close();  // in parallel we need to explicitly close the show here while MPI is still valid.

  if( plotOption )
  {
    psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,false);
    psp.set(GI_TOP_LABEL,"final grid");  // set title
    PlotIt::plot(ps,cg[(numberOfSteps%2)],psp);
  }

  // clean up :
  for( int g=0; g<numberOfGridsToMove; g++ )
  {
    delete transform0[g];
    delete transform1[g];
  }


  Overture::finish();
  return 0;
}

```

3.12 Example: Adaptive Grids

This example shows how to build an adaptive grid with a refinement level. Starting from a grid collection consisting of a single square, two refinement grids are added. Grid functions are created and the grid functions are plotted on the adaptive grid and the refinement levels.

(file `Overture/primer/amrExample1.C`)

```
#include "Overture.h"
#include "SquareMapping.h"
#include "AnnulusMapping.h"
#include "HDF_DataBase.h"
#include "PlotStuff.h"
#include "display.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" --------------------------------------------------------------------------- \n");
  printf(" Demonstrate the method for adding refinement levels to a grid collection    \n");
  printf(" Show how to plot the grid collection, refinement levels, or grid functions \n");
  printf(" --------------------------------------------------------------------------- \n");


  PlotStuff ps;               // for plotting
  PlotStuffParameters psp;

  SquareMapping mapping(-1., 1., -1., 1.);            // Create a SquareMapping
  mapping.setGridDimensions(axis1,11); mapping.setGridDimensions(axis2,11);
//AnnulusMapping mapping;            // Create an Annulus
//mapping.setGridDimensions(axis1,21); mapping.setGridDimensions(axis2,11);

  MappedGrid mg(mapping);      // grid for a mapping
  mg.update();

  //  Create a two-dimensional GridCollection with one grid.
  GridCollection gc(2,1);
  gc[0].reference(mg);
  gc.updateReferences();
  gc.update(MappedGrid::THEvertex);

  psp.set(GI_TOP_LABEL,"initial grid");  // set title
  PlotIt::plot(ps,gc,psp);                       // plot the grid

  // assign values to the grid collection function
  realGridCollectionFunction u(gc);
  Index I1,I2,I3;
  int grid;
  for( grid=0; grid<gc.numberOfComponentGrids(); grid++ )
  {
    getIndex(gc[grid].dimension(),I1,I2,I3);
    u[grid](I1,I2,I3)=sin(gc[grid].vertex()(I1,I2,I3,axis1)*Pi)*sin(gc[grid].vertex()(I1,I2,I3,axis2)*Pi);
  }
  ps.erase();
  psp.set(GI_TOP_LABEL,"u on initial grid");
  PlotIt::contour(ps,u,psp);

  gc.update(GridCollection::THErefinementLevel);  // indicate that we are want a refinement level

  // Add a refinement, specify position in the coarse grid index space
  IntegerArray range(2,3), factor(3);
  range(0,0) = 2; range(1,0) = 6;
  range(0,1) = 2; range(1,1) = 6;
  range(0,2) = 0; range(1,2) =  0;
  factor = 4;                            // refinement factor = 4
  Integer level = 1;
  grid = 0;                              // refine this base grid
  gc.addRefinement(range, factor, level, grid);    // add a refinement grid to level 1

  range(0,0) = 4; range(1,0) = 8;
  range(0,1) = 4; range(1,1) = 8;
```

```
  range(0,2) = 0; range(1,2) = 0;
  gc.addRefinement(range, factor, level, grid);   // add another refinement grid to level 1

  gc.update(GridCollection::THErefinementLevel);

  gc.refinementFactor.display("gc.refinementFactor");
  gc.refinementLevel[0].refinementFactor.display("gc.refinementLevel[0].refinementFactor");
  gc.refinementLevel[1].refinementFactor.display("gc.refinementLevel[1].refinementFactor");

  ps.erase();
  psp.set(GI_TOP_LABEL,"refined grid");
  PlotIt::plot(ps,gc,psp);                        // plot the grid collection including refinements

  ps.erase();
  psp.set(GI_TOP_LABEL,"refinementLevel[0]");
  PlotIt::plot(ps,gc.refinementLevel[0],psp);     // plot refinement level 0 only

  u.updateToMatchGrid(gc);       // tell u that the gridCollection has been changed

  gc.update(MappedGrid::THEvertex);

  gc.setMaskAtRefinements();

  for( grid=0; grid<gc.numberOfComponentGrids(); grid++ )
  {
    printf(" grid=%i, level=%i, refinementFactor=%i\n",grid,gc.refinementLevelNumber(grid),
             gc.refinementFactor(axis1,grid));


    getIndex(gc[grid].dimension(),I1,I2,I3);
    u[grid](I1,I2,I3)=sin(gc[grid].vertex()(I1,I2,I3,axis1)*Pi)*sin(gc[grid].vertex()(I1,I2,I3,axis2)*Pi);

  }
  ps.erase();
  psp.set(GI_TOP_LABEL,"u on refined grid");
  PlotIt::contour(ps,u,psp);

  ps.erase();
  psp.set(GI_TOP_LABEL,"refinementLevel[1]");
  PlotIt::plot(ps,gc.refinementLevel[1],psp);     // plot refinement level 1 only

  // Now plot the refinementLevel's in the grid functions
  ps.erase();
  psp.set(GI_TOP_LABEL,"u.refinementLevel[0]");
  PlotIt::contour(ps,u.refinementLevel[0],psp);   // plot u on refinement level 0

  ps.erase();
  psp.set(GI_TOP_LABEL,"u.refinementLevel[1]");
  PlotIt::contour(ps,u.refinementLevel[1],psp);    // plot u on refinement level 1

  Overture::finish();
  return 0;
}
```

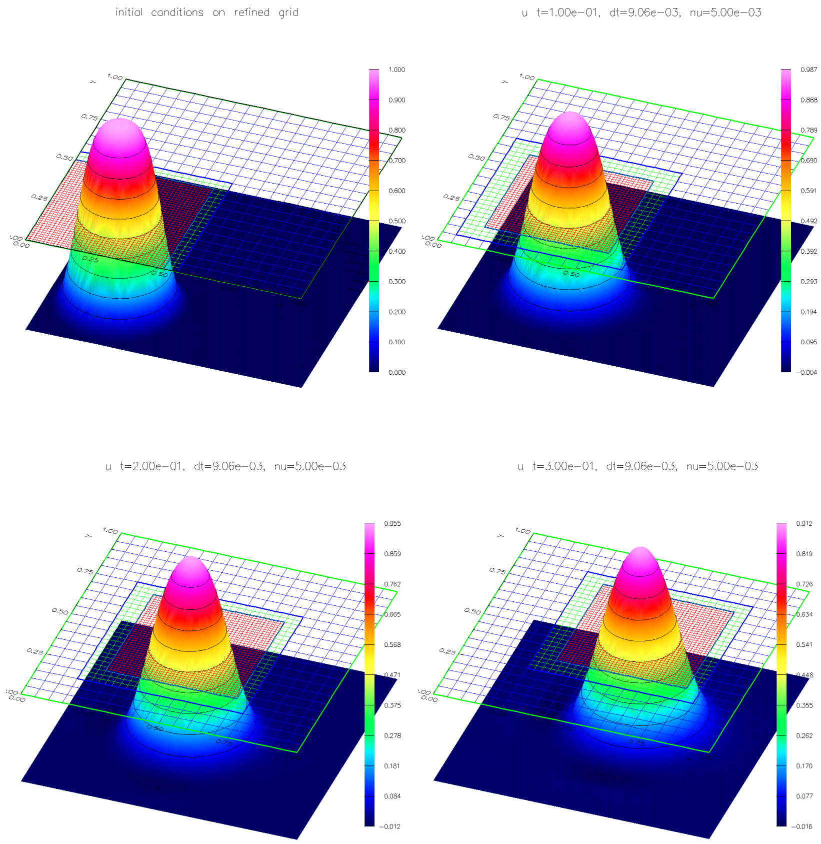


Figure 10: Results from amrh, solving a convection diffusion equation with adaptive mesh refinement.

3.13 Example: Time dependent adaptive mesh refinement solver: amrh

The file `Overture/primer/amrh.C` solves a convection diffusion equation using adaptive mesh refinement (AMR). The equation

$$u_t + au_x + bu_y = \nu \Delta u$$

is advanced with a fouth-order accurate Runge-Kutta time stepping algorithm. Every few steps a new AMR grid is computed, based on an estimate of the error. The solution must then be interpolated from the old AMR grid to the new AMR grid. For simplicity, a single time step is used on all grids. The AMR algorithm is implemented with the help of the following classes

ErrorEstimator : class used to compute error estimates.

Regrid : class used to build a new AMR grid.

InterpolateRefinements : used to

- interpolate from one AMR grid to a second AMR grid,
- interpolate ghost-boundaries of refinement grids
- interpolate coarse grid ppoints that are hidden by refinement grids.

This class in turn uses the **Interpolate** class which knows how to interpolate refinement patch points from a coarser grid.

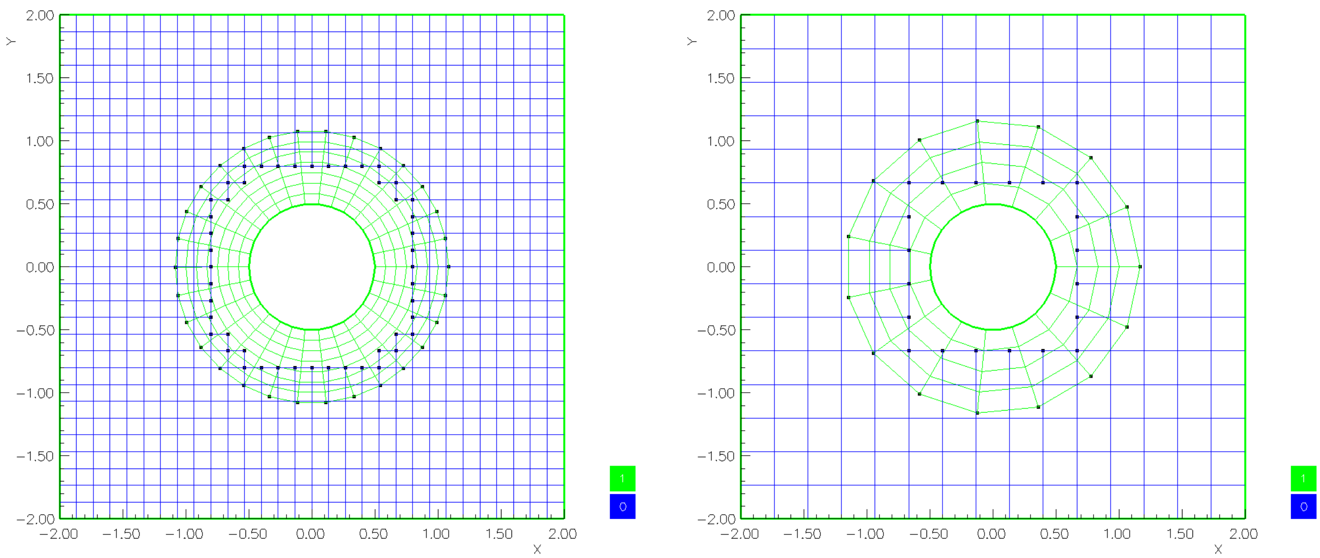
See the documents [2],[**?**] for further details.

3.14 Example: Multigrid Overlapping Grids

This example shows how to manipulate an overlapping grid that has more than one multigrid level. The overlapping grid should be created with more than one multigrid level as shown for example in the `cicmg.cmd` command file presented in the `ogen` grid generator documentation.

The following program reads in the overlapping grid. It then plots the overlapping grids at the different levels (each multigrid level is a valid `CompositeGrid`). Next a grid function is built and the different multigrid levels of the grid function are assigned and plotted. (file `Overture/primer/mgExample1.C`)

```
#include "Overture.h"
#include "PlotStuff.h"

int
main(int argc, char* argv[])
{
  ios::sync_with_stdio();
  Index::setBoundsCheck(On);

  printf(" --------------------------------------------------------------------------- \n");
  printf("     Demonstrate how to use multigrid levels with an overlapping grid.       \n");
  printf(" The overlapping grid should be created with more than 1 multigrid level,    \n");
  printf(" see the cicmg.cmd command file as an example.                               \n");
  printf(" --------------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "mgExample1>> Enter the name of the (old) overlapping grid file: (cicmg for example)" << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

  PlotStuff ps;               // for plotting
  PlotStuffParameters psp;
  char buff[80];              // buffer for sPrintF

  // plot each multigrid level (the grid plotter also knows how to plot multigrid levels,
  //   if we were to use: PlotIt::plot(ps,cg); and choose the menu option "plot a multigrid level" )
  int grid,level;
  for( level=0; level<cg.numberOfMultigridLevels(); level++ )
  {
    psp.set(GI_TOP_LABEL,sPrintF(buff,"Multigrid level %i",level));  // set title
    PlotIt::plot(ps,cg.multigridLevel[level],psp);    // plot the CompositeGrid for a given level
  }

  // create a grid function on the multigrid-overlapping grid and assign values to it
  realCompositeGridFunction u(cg);
  Index I1,I2,I3;
  for( level=0; level<cg.numberOfMultigridLevels(); level++ )
  {
    CompositeGrid & cgLevel = cg.multigridLevel[level];            // make a reference to a given level
    realCompositeGridFunction & uLevel = u.multigridLevel[level];
    for( grid=0; grid<cgLevel.numberOfComponentGrids(); grid++ )
    {
      getIndex(cgLevel[grid].dimension(),I1,I2,I3);
      uLevel[grid](I1,I2,I3)=sin(cgLevel[grid].vertex()(I1,I2,I3,axis1)*Pi)    // u = sin(x*Pi)*sin(y*Pi)
                            *sin(cgLevel[grid].vertex()(I1,I2,I3,axis2)*Pi);
    }
    psp.set(GI_TOP_LABEL,sPrintF(buff,"u on multigrid level %i",level));  // set title
    PlotIt::contour(ps,uLevel,psp);
  }

  return 0;
}
```



(a)

(b)

Figure 11: The two multigrid levels for the grid created by `cicmg.cmd`. A `CompositeGrid cg` can hold multiple multigrid levels which are referenced as `cg.multigridLevel[level]`, `level=0,1,...`. Each `cg.multigridLevel[level]` is itself a valid `CompositeGrid`.

3.15 Example: Solving elliptic problems on each multigrid level

This example shows how to solve an elliptic problem on each multigrid level of an overlapping grid. The example also shows how to access the matrix coefficients and compute the residual. The overlapping grid should be created with more than one multigrid level as shown for example in the `cicmg.cmd` command file (see the previous section).

The following program reads in the overlapping grid. Grid functions are built on the multigrid overlapping grid to hold the solution, right-hand-side and the coefficient matrix. Since the `CompositeGridOperators` and the sparse solver class `Oges` do not know about multigrid levels it is necessary to explicitly build these objects for each level. For each multigrid level we build a coefficient matrix and solve a problem. The errors are printed. (file `Overture/primer/mgExample2.C`)

```
#include "Overture.h"
#include "CompositeGridOperators.h"
#include "Oges.h"
#include "OGPolyFunction.h"


// These macros define how to access the elements in a coefficient matrix. See the example below
#undef C
#undef M123
#define M123(m1,m2,m3) (m1+halfWidth1+width1*(m2+halfWidth2+width2*(m3+halfWidth3)))
#define COEFF(m1,m2,m3,I1,I2,I3) c(M123(m1,m2,m3),I1,I2,I3)

int
main(int argc, char* argv[])
{
  ios::sync_with_stdio();
  Index::setBoundsCheck(On);

  printf(" -------------------------------------------------------------------------- \n");
  printf(" Demonstrate how to solve an elliptic problem on different multigrid levels.\n");
  printf(" The overlapping grid should be created with more than 1 multigrid level,   \n");
  printf(" see the cicmg.cmd command file as an example.                              \n");
  printf(" -------------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "mgExample2>> Enter the name of the (old) overlapping grid file: (cicmg for example)" << endl;
  cin >> nameOfOGFile;

  // create and read in a (multigrid) CompositeGrid
  CompositeGrid cgmg;
  getFromADataBase(cgmg,nameOfOGFile);
  cgmg.update();

  // allocate operators and sparse solvers for all levels
  CompositeGridOperators *opMG = new CompositeGridOperators [cgmg.numberOfMultigridLevels()];
  Oges *solverMG = new Oges [cgmg.numberOfMultigridLevels()];
  // Now build a coefficient matrix
  Range all;
  const int stencilSize=int( pow(3,cgmg.numberOfDimensions())+1.5 );
  realCompositeGridFunction coeffMG(cgmg,stencilSize,all,all,all);

  // build grid functions for the solution and rhs
  realCompositeGridFunction uMG(cgmg), fMG(cgmg);
  // create a twilight-zone function for checking the errors
  int degreeOfSpacePolynomial = 2;
  int degreeOfTimePolynomial = 1;
  int numberOfComponents = cgmg.numberOfDimensions();
  OGPolyFunction exact(degreeOfSpacePolynomial,cgmg.numberOfDimensions(),numberOfComponents,
                       degreeOfTimePolynomial);

  // Now loop over each level. Solve a Poisson problem on each level
  for( int level=0; level<cgmg.numberOfMultigridLevels(); level++ )
  {
    // first make some references for ease of use
    CompositeGrid & cg = cgmg.multigridLevel[level];
    if( FALSE && level>0 )
    {
      cg.interpoleeGrid[0].display("cg.interpoleeGrid");
      cg.interpolationPoint[0].display("cg.interpolationPoint.display");
    }
```

```



    realCompositeGridFunction & coeff = coeffMG.multigridLevel[level];
    realCompositeGridFunction & u = uMG.multigridLevel[level];
    realCompositeGridFunction & f = fMG.multigridLevel[level];
    CompositeGridOperators & op = opMG[level];

    op.updateToMatchGrid(cg);  // the operators on this level must be associated with a grid (once only)
    op.setStencilSize(stencilSize);   // set stencil size for operators

    coeff.setIsACoefficientMatrix(TRUE, stencilSize);
    coeff.setOperators(op);
    coeff = op.laplacianCoefficients();
    // fill in the coefficients for the boundary conditions
    coeff.applyBoundaryConditionCoefficients(0,0,BCTypes::dirichlet,  BCTypes::allBoundaries);
    coeff.applyBoundaryConditionCoefficients(0,0,BCTypes::extrapolate,BCTypes::allBoundaries);
    coeff.finishBoundaryConditions();

    Oges & solver = solverMG[level];
    solver.setCoefficientArray( coeff );   // supply coefficients
    solver.updateToMatchGrid( cg );        // create a solver, and update to the grid (once only)

    // assign the rhs: Laplacian(u)=f, u=exact on the boundary
    Index I1,I2,I3, Ia1,Ia2,Ia3;
    int side,axis,grid;
    Index Ib1,Ib2,Ib3;
    for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
    {
      MappedGrid & mg = cg[grid];
      getIndex(mg.indexRange(),I1,I2,I3);
      if( cg.numberOfDimensions()==1 )
        f[grid](I1,I2,I3)=exact.xx(mg,I1,I2,I3,0);
      else if( cg.numberOfDimensions()==2 )
        f[grid](I1,I2,I3)=exact.xx(mg,I1,I2,I3,0)+exact.yy(mg,I1,I2,I3,0);
      else
        f[grid](I1,I2,I3)=exact.xx(mg,I1,I2,I3,0)+exact.yy(mg,I1,I2,I3,0)+exact.zz(mg,I1,I2,I3,0);
      // loop over the boundaries
      for( axis=0; axis<mg.numberOfDimensions(); axis++ )
      for( side=0; side<=1; side++ )
      {
        if( mg.boundaryCondition()(side,axis) > 0 )
        {
          getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
          f[grid](Ib1,Ib2,Ib3)=exact(mg,Ib1,Ib2,Ib3,0);
        }
      }
    }

    u=0.;  // initial guess for iterative solvers
    real time0=getCPU();
    solver.solve( u,f );   // solve the equations
    real time=getCPU()-time0;
    cout << "level=" << level << ", time for solve of the Dirichlet problem = " << time << endl;

    real error=0.;
    for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
    {
      getIndex(cg[grid].indexRange(),I1,I2,I3);
      where( cg[grid].mask()(I1,I2,I3)!=0 )
        error=max(error,max(abs(u[grid](I1,I2,I3)-exact(cg[grid],I1,I2,I3,0))));
    }
    printf("level=%i, Maximum error with dirichlet bc's= %e\n",level,error);

    // Now compute the maximum residual
    real maximumResidual=0.;
    realCompositeGridFunction residual(cg);
    // These stencil widths are used by the COEFF macro
    const int width1=3, halfWidth1=width1/2, width2=3, halfWidth2=width2/2;
    const int width3= cg.numberOfDimensions()==2 ? 1 : 3, halfWidth3=width3/2;

    for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
    {
```

```
    realArray & c = coeff[grid];
    realArray & uu= u[grid];
    realArray & ff= f[grid];
    realArray & res = residual[grid];
    // We must first reshape the arrays so that we can multiply by the coefficient matrix
    uu.reshape(1,uu.dimension(0),uu.dimension(1),uu.dimension(2));
    ff.reshape(1,ff.dimension(0),ff.dimension(1),ff.dimension(2));
    res.reshape(1,res.dimension(0),res.dimension(1),res.dimension(2));

    getIndex(cg[grid].indexRange(),I1,I2,I3);

    if( cg.numberOfDimensions()==2 )
    {
      // The COEFF macro makes the coeff array look like a 6 dimensional array.
      res(0,I1,I2,I3)=ff(0,I1,I2,I3)-(
         COEFF( 0, 0,0,I1,I2,I3)*uu(0,I1  ,I2  ,I3)
        +COEFF( 1, 0,0,I1,I2,I3)*uu(0,I1+1,I2  ,I3)
        +COEFF( 0, 1,0,I1,I2,I3)*uu(0,I1  ,I2+1,I3)
        +COEFF(-1, 0,0,I1,I2,I3)*uu(0,I1-1,I2  ,I3)
        +COEFF( 0,-1,0,I1,I2,I3)*uu(0,I1  ,I2-1,I3)
        +COEFF( 1, 1,0,I1,I2,I3)*uu(0,I1+1,I2+1,I3)
        +COEFF( 1,-1,0,I1,I2,I3)*uu(0,I1+1,I2-1,I3)
        +COEFF(-1, 1,0,I1,I2,I3)*uu(0,I1-1,I2+1,I3)
        +COEFF(-1,-1,0,I1,I2,I3)*uu(0,I1-1,I2-1,I3)
        );
    }
    else
    {
      res(0,I1,I2,I3)=ff(0,I1,I2,I3)-(
         COEFF(-1,-1,-1,I1,I2,I3)*uu(0,I1-1,I2-1,I3-1)
        +COEFF( 0,-1,-1,I1,I2,I3)*uu(0,I1  ,I2-1,I3-1)
        +COEFF( 1,-1,-1,I1,I2,I3)*uu(0,I1+1,I2-1,I3-1)
        +COEFF(-1, 0,-1,I1,I2,I3)*uu(0,I1-1,I2  ,I3-1)
        +COEFF( 0, 0,-1,I1,I2,I3)*uu(0,I1  ,I2  ,I3-1)
        +COEFF( 1, 0,-1,I1,I2,I3)*uu(0,I1+1,I2  ,I3-1)
        +COEFF(-1, 1,-1,I1,I2,I3)*uu(0,I1-1,I2+1,I3-1)
        +COEFF( 0, 1,-1,I1,I2,I3)*uu(0,I1  ,I2+1,I3-1)
        +COEFF( 1, 1,-1,I1,I2,I3)*uu(0,I1+1,I2+1,I3-1)

        +COEFF(-1,-1, 0,I1,I2,I3)*uu(0,I1-1,I2-1,I3  )
        +COEFF( 0,-1, 0,I1,I2,I3)*uu(0,I1  ,I2-1,I3  )
        +COEFF( 1,-1, 0,I1,I2,I3)*uu(0,I1+1,I2-1,I3  )
        +COEFF(-1, 0, 0,I1,I2,I3)*uu(0,I1-1,I2  ,I3  )
        +COEFF( 0, 0, 0,I1,I2,I3)*uu(0,I1  ,I2  ,I3  )
        +COEFF( 1, 0, 0,I1,I2,I3)*uu(0,I1+1,I2  ,I3  )
        +COEFF(-1, 1, 0,I1,I2,I3)*uu(0,I1-1,I2+1,I3  )
        +COEFF( 0, 1, 0,I1,I2,I3)*uu(0,I1  ,I2+1,I3  )
        +COEFF( 1, 1, 0,I1,I2,I3)*uu(0,I1+1,I2+1,I3  )

        +COEFF(-1,-1, 1,I1,I2,I3)*uu(0,I1-1,I2-1,I3+1)
        +COEFF( 0,-1, 1,I1,I2,I3)*uu(0,I1  ,I2-1,I3+1)
        +COEFF( 1,-1, 1,I1,I2,I3)*uu(0,I1+1,I2-1,I3+1)
        +COEFF(-1, 0, 1,I1,I2,I3)*uu(0,I1-1,I2  ,I3+1)
        +COEFF( 0, 0, 1,I1,I2,I3)*uu(0,I1  ,I2  ,I3+1)
        +COEFF( 1, 0, 1,I1,I2,I3)*uu(0,I1+1,I2  ,I3+1)
        +COEFF(-1, 1, 1,I1,I2,I3)*uu(0,I1-1,I2+1,I3+1)
        +COEFF( 0, 1, 1,I1,I2,I3)*uu(0,I1  ,I2+1,I3+1)
        +COEFF( 1, 1, 1,I1,I2,I3)*uu(0,I1+1,I2+1,I3+1)
        );
    }
    // reshape the arrays back to their original shape -- this would
    // be essential if the GridFunctions u,f or res were used again
    uu.reshape(uu.dimension(1),uu.dimension(2),uu.dimension(3));
    ff.reshape(ff.dimension(1),ff.dimension(2),ff.dimension(3));
    res.reshape(res.dimension(1),res.dimension(2),res.dimension(3));

    // compute the residual at all discretization points
    where( cg[grid].mask()(I1,I2,I3) > 0 )
      maximumResidual=max(maximumResidual,max(fabs(res(I1,I2,I3))));
  }
  printf("level=%i, Maximum residual with dirichlet bc's= %e\n",level,maximumResidual);

```

```
  }
  return 0;
}
```

For the grid `cicmg.hdf` the output is

```
ultrabert{henshaw}74: mgExample2
A++ Internal_Index bounds checking: ON
 ---------------------------------------------------------------------------
 Demonstrate how to solve an elliptic problem on different multigrid levels.
 The overlapping grid should be created with more than 1 multigrid level,
 see the cicmg.cmd command file as an example.
 ---------------------------------------------------------------------------
mgExample2>> Enter the name of the (old) overlapping grid file: (cicmg for example)
../cgsh/cicmg
getFromADataBase: number of CompositeGrid(s) found =1, name[0]=cic
>>>>>> SparseRep::update to match grid <<<<<<<<<<
>>>>>> SparseRep::update to match grid <<<<<<<<<<
level=0, time for solve of the Dirichlet problem = 0.165335
level=0, Maximum error with dirichlet bc's= 2.058983e-03
level=0, Maximum residual with dirichlet bc's= 4.582405e-04
>>>>>> SparseRep::update to match grid <<<<<<<<<<
>>>>>> SparseRep::update to match grid <<<<<<<<<<
level=1, time for solve of the Dirichlet problem = 0.0532484
level=1, Maximum error with dirichlet bc's= 6.177902e-03
level=1, Maximum residual with dirichlet bc's= 1.449585e-04
ultrabert{henshaw}75:
```

3.16 Example: Calling a fortran function for each component grid.

In this example we solve the equation $u_t = f(u, \mathbf{x}, t)$ on an overlapping grid. A Fortran function is called to compute $f(u, \mathbf{x}, t)$ for each component grid (file `Overture/primer/callingFortran.C`)

```
#include "Overture.h"
#include "Ogshow.h"
#include "CompositeGridOperators.h"
#include "PlotStuff.h"
#include "display.h"

// fortran names may have an appended underscore which is added by the EXTERN_C_NAME macro:
#define mySolver EXTERN_C_NAME(mysolver)

extern "C"
{
  // fortran variables are passed by reference:
  void mySolver( const real &t, const real &dt,const real &a,const real &b,const real &nu, const int&nd,
     const int &nd1a,const int &nd1b,const int &nd2a,const int &nd2b,const int &nd3a,const int &nd3b,
     const int &n1a,const int &n1b,const int &n2a,const int &n2b,const int &n3a,const int &n3b,
               const real &x,const real &u, real &dudt );
}

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------------------- \n");
  printf(" Solve a PDE u.t=f(u,x,t). Call a fortran routine to compute f(u,x,t) for     \n");
  printf(" each component grid. Plot the results.                                      \n");
  printf(" ------------------------------------------------------------------------- \n");

  aString nameOfOGFile;
  cout << "callingFortran>> Enter the name of the (old) overlapping grid file:" << endl;
  cin >> nameOfOGFile;

  // create and read in a CompositeGrid
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update(MappedGrid::THEvertex | MappedGrid::THEmask);      // build vertices and mask

  // Interpolant interpolant(cg);                                 // Make an interpolant
  Interpolant & interpolant = *new Interpolant(cg);               // do this instead for now.

  CompositeGridOperators operators(cg);                        // operators for a CompositeGrid

  Range all;
  realCompositeGridFunction u(cg,all,all,all,1);               // create grid functions
  realCompositeGridFunction dudt(cg,all,all,all,1);
  u.setOperators(operators);
  u.setName("u");                                              // name the grid function

  u=1.;                                                        // initial condition
  dudt=0.;

  real t=0, dt=.01;                                           // initialize time and time step
  real a=1., b=1., nu=.1;                                     // initialize parameters

  bool openGraphicsWindow=TRUE;
  PlotStuff ps(openGraphicsWindow,"callingFortran");
  PlotStuffParameters psp;


  aString buff;                                              // buffer for sPrintF
  int numberOfTimeSteps=200;
  for( int i=0; i<numberOfTimeSteps; i++ )                    // take some time steps
  {
    if( (i % 10)==0 )
    {
      psp.set(GI_TOP_LABEL,sPrintF(buff,"Solution t=%f",t));  // set title
      ps.erase();
```

```
      PlotIt::contour(ps,u,psp);
      ps.redraw(true);
      psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);     // set this to run in "movie" mode (after first plot)
    }

    int grid;
    for( grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over grids
    {
      MappedGrid & mg = cg[grid];
      realArray & ug = u[grid];
      realArray & dudtg = dudt[grid];
      realArray & x = mg.vertex();  // array of vertices

      const IntegerArray & d = mg.dimension();
      const IntegerArray & gir= mg.gridIndexRange();
      const int nd=cg.numberOfDimensions();

      // call a fortran function to compute du/dt
      // (This function does not currently solve the convection diffusion equation)
      mySolver( t,dt,a,b,nu,nd, d(0,0),d(1,0),d(0,1),d(1,1),d(0,2),d(1,2),
              gir(0,0),gir(1,0),gir(0,1),gir(1,1),gir(0,2),gir(1,2),
              *x.getDataPointer(),*ug.getDataPointer(), *dudtg.getDataPointer() );

      // display(ug,"ug","%6.3f");
      // display(dudtg,"dudtg","%6.3f");

      ug+=dt*dudtg;
    }
    t+=dt;
    u.interpolate();                                           // interpolate
    // u.display("u after interpolate");

    // apply a dirichlet BC on all boundaries:
    u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
    u.applyBoundaryCondition(0,BCTypes::extrapolate,BCTypes::allBoundaries,0.);
    u.finishBoundaryConditions();
  }

  psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,false);
  psp.set(GI_TOP_LABEL,sPrintF(buff,"Solution t=%f",t));  // set title
  ps.erase();
  PlotIt::contour(ps,u,psp);

  Overture::finish();
  return 0;

}
```

The fortran function is defined in the file `Overture/primer/mySolver.f`

3.17 Example: Building an overlapping grid directly in a program.

In this example we generate an overlapping grid directly in a program, rather than interactively, (file `Overture/primer/gridGenExample.C`)

```
#include "Overture.h"
#include "SquareMapping.h"
#include "AnnulusMapping.h"
#include "Ogen.h"
#include "PlotStuff.h"
#include "StretchTransform.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture

  printf(" ------------------------------------------------------------------------- \n");
  printf(" This example shows how to build an overlapping grid directly in a program, \n");
  printf(" rather than interactively.                                                 \n");
  printf(" ------------------------------------------------------------------------- \n");

  int plotOption=true; // set to false for no plotting
  PlotStuff gi(plotOption,"GridGenExample");
  PlotStuffParameters gip;

  // By default start saving the command file called "motion.cmd"
  aString logFile="gridGen.cmd";
  gi.saveCommandFile(logFile);
  cout << "User commands are being saved in the file `" << (const char *)logFile << "'\n";

  // First build mappings
  SquareMapping map1(-1., 1., -1., 1.);        // Create a square
  map1.setGridDimensions(axis1,21); map1.setGridDimensions(axis2,21);

  AnnulusMapping map2;            // Create an Annulus
  map2.setRadii(.3,.6);
  map2.setGridDimensions(axis1,41); map2.setGridDimensions(axis2,9);
  // set outer boundary to be interpolation:
  map2.setBoundaryCondition(End,axis2,0);

  MappingInformation mapInfo;
  mapInfo.graphXInterface=&gi;

  // stretch the annulus
  StretchTransform map3;
  map3.setMapping(map2);

  // Here we show how to use interactive commands to set the properties of the
  // stretching (rather than calling member functions to set the properties).

  // Make a list of interactive commands:
  aString cmds[5] = {"Stretch r2:itanh",
                     "STP:stretch r2 itanh: position and min dx 0 0.02",
                     "stretch grid",
                     "exit",
                     "" }; // N.B. string should be null terminated
  // Tell the graphics interface to use these commands when requested
  gi.readCommandsFromStrings(cmds);
  // "interactively" update the StretchTransform (which will read the command strings)
  map3.update(mapInfo);

  // map3.update(mapInfo);


  // Put the mappings into a list
  mapInfo.mappingList.addElement(map1);
  mapInfo.mappingList.addElement(map3);

  // Indicate which mappings should be used in the overlapping grid
  const int numberOfDimensions=2, numberOfGrids=2;
  IntegerArray mapList(numberOfGrids);
```

```
  mapList(0)=0;
  mapList(1)=1;

  // build an empty overlapping grid
  CompositeGrid cg;

  // Create an overlapping grid generator
  Ogen ogen(gi);

  // Put the mappings into the CompositeGrid
  ogen.buildACompositeGrid(cg,mapInfo,mapList);

  //  ogen.debug=3; // turn this on to show intermediate results

  // ** generate the overlapping grid**
  ogen.updateOverlap(cg);

  // Plot the overlapping grid.
  gi.erase();
  gip.set(GI_TOP_LABEL,"Grid after Ogen");  // set title
  PlotIt::plot(gi,cg,gip);                       // plot the grid


  Overture::finish();
  return 0;
}
```

4 Single versus double precision

Overture is designed so that one writes a single code that can be used in either single or double precision. The types `real`, `realMappedGridFunction`, `realCompositeGridFunction` etc. are either `float` or `double` depending on whether the Overture library has been built with the double precision option. Note that the Overture library must be entirely recompiled for double precision. Thus the steps to take to run in **double precision** are

1. Build the Overture library in double precision.
2. remake any overlapping grids with the new double precision version of the grid generator.

Thus one would never explicitly create a `doubleMappedGridFunction` (except in very special cases). The file `Overture/include/OvertureDefine.h` will define the macro `OV_USE_DOUBLE` if Overture has been compiled in double precision.

5 Makefile's and .cshrc files

When Overture is installed using the `Overture/configure` script `Makefile`'s are build in sub-directories of Overture. You can copy the `Makefile` from the `Overture/primer` directory to use as a starting point for making other applications.

The Overture `Makefile`'s require a number of environmental variables to be defined so that it knows where to find various libraries. Here are some example `.cshrc` files that could be used to define these environmental variables. **NOTE** that if you are making (i.e. installing) Overture (as opposed to just using it) you must explicitly define the locations of OpenGL, HDF and APlusPlus (rather than using the relative paths given below).
Note: If you have made the **double precision** version of Overture then you should add `-DDOUBLE` to the `CCFLAGS` definitions in the Makefiles below as in `CCFLAGS= -DDOUBLE -O ...`

5.1 Sun Workstations

Here is a `.cshrc` file and a `Makefile` that can be used on a Sun workstation.
File `Overture/primer/cshrc.sun`:

```
#
# a sample .cshrc to use with Overture on a sun
#   NOTE: If you are making Overture (as opposed to just using it) you must explicitly
#   define the locations of OpenGL, HDF and APlusPlus (rather than using the relative path below)
#
setenv XLIBS      /usr/openwin
setenv MOTIF      /usr/dt
setenv Overture   /n/c19s3/Overture/Overture.v12
# use full path name for next three variables if you are installing Overture
setenv OpenGL     $Overture/OpenGL
setenv HDF        $Overture/HDF
setenv APlusPlus  $Overture/A++
# this next variable tells the run time loader where to find dynamic (.so) libraries
setenv LD_LIBRARY_PATH  ${MOTIF}/lib:${XLIBS}/lib:${HDF}/lib:${OpenGL}/lib:$Overture/lib:${APlusPlus}
```

File `Overture/primer/Makefile.sun`:

```
#
# Makefile for a sun (to be combined with .cshrc.sun)
# assumes the existence of the file mappedGridExample1.C (which can be copied from $Overture/primer)
#
INCLUDE = -I. -I$(Overture)/include  -I$(A++)/include -I$(OpenGL)/include
CLIBS= -L$(Overture)/lib -lOverture_static -lOverture -lOverture_static -L$(A++) -lA++ -lA++_static -lA++
GLIBS= -L$(OpenGL)/lib -lMesaaux -lMesatk -lMesaGLU -lMesaGL -lGLw -L$(HDF)/lib -lmfhdf -ldf -ljpeg -lz -L$(MOTIF)/lib -lXm \
       -L$(XLIBS)/lib -lXt -lX11 -lXext -lF77 -lM77 -lV77 -lsunmath -lm

cc= cc
CC=       CC
FC=       f77
FFLAGS =   -g
CCFLAGS=  -O  $(INCLUDE)

.SUFFIXES:
.SUFFIXES:.f .o .C .o .c .o
.C.o:; $(CC) $(CCFLAGS) -c $*.C
.c.o:; $(cc) $(ccFLAGS) -c $*.c
.f.o:; $(FC) $(FFLAGS) -c $*.f

mappedGridExample1 = mappedGridExample1.o
mappedGridExample1: $(mappedGridExample1)
        $(CC) $(CCFLAGS) -o mappedGridExample1 $(mappedGridExample1) $(CLIBS) $(GLIBS) -lm

```

5.2 SGI Workstations

Here is a `.cshrc` file and a `Makefile` that can be used on an SGI machine.
File `Overture/primer/cshrc.sgi64`:

```
#
# a sample .cshrc to use with Overture on an sgi
#   NOTE: If you are making Overture (as opposed to just using it) you must explicitly
#   define the locations of OpenGL, HDF and APlusPlus (rather than using the relative path below)
```

```
#
setenv XLIBS  /usr
setenv MOTIF /usr
setenv Overture /usr/src/Overture/Overture.v12
# use full path name for next three variables if you are installing Overture
setenv OpenGL     $Overture/OpenGL
setenv HDF        $Overture/HDF
setenv APlusPlus  $Overture/A++
# this next variable tells the run time loader where to find dynamic (.so) libraries
setenv LD_LIBRARY_PATH  $HDF/lib:$Overture/lib:${APlusPlus}
```

File `Overture/primer/Makefile.sgi64`:

```
#
# Makefile for irix64 (to be combined with .cshrc.sgi64)
# assumes the existence of the file mappedGridExample1.C (which can be copied from $Overture/primer)
#

INCLUDE = -I. -I$(Overture)/include  -I$(A++)/include
LIBAPP=         -L$(A++) -lA++ -lA++_static -lA++
CLIBS= -L$(Overture)/lib -lOverture_static -lOverture -lOverture_static -L$(A++) -lA++ -lA++_static -lA++
GLIBS= -L$(OpenGL)/lib64 -lGLw -lGL -lGLU  -lGLw -L$(HDF)/lib -lmfhdf -ldf -ljpeg -lz -L$(MOTIF)/lib64 -lXm -L$(XLIBS)/lib64 -l

cc= cc
CC=       CC
FC=       f77
FFLAGS =   -g
CCFLAGS=  -64 -mips4 -woff 1188,1047,1681,1021,1110 -Wl,-woff,85 $(INCLUDE)

.SUFFIXES:
.SUFFIXES:.f .o .C .o .c .o
.C.o:; $(CC) $(CCFLAGS) -c $*.C
.c.o:; $(cc) $(ccFLAGS) -c $*.c
.f.o:; $(FC) $(FFLAGS) -c $*.f

mappedGridExample1 = mappedGridExample1.o
mappedGridExample1: $(mappedGridExample1)
        $(CC) $(CCFLAGS) -o mappedGridExample1 mappedGridExample1.o $(CLIBS) $(GLIBS) -lm
```

5.3 Pentium with Linux

File `Overture/primer/Makefile.linux`:

```
#
# Makefile for a linux pentium box
#
INCLUDE = -I. -I$(Overture)/include  -I$(A++)/include -I$(OpenGL)/include
CLIBS= -L$(Overture)/lib -lOverture_static -lOverture -lOverture_static -L$(A++) -lA++ -lA++_static -lA++
GLIBS= -L$(OpenGL)/lib -lMesaaux -lMesatk -lMesaGLU -lMesaGL -lGLw -L$(HDF)/lib -lmfhdf -ldf -ljpeg -lz -L$(MOTIF)/lib -lXm \
       -L$(XLIBS)/lib -lXt -lX11 -lXext -lXpm -L/usr/lib -lf2c -lgcc -lc -lm

cc= gcc
CC=       g++
FC=       g77
FFLAGS =   -g
CCFLAGS=  -O  $(INCLUDE)

.SUFFIXES:
.SUFFIXES:.f .o .C .o .c .o
.C.o:; $(CC) $(CCFLAGS) -c $*.C
.c.o:; $(cc) $(ccFLAGS) -c $*.c
.f.o:; $(FC) $(FFLAGS) -c $*.f

mappedGridExample1 = mappedGridExample1.o
mappedGridExample1: $(mappedGridExample1)
        $(CC) $(CCFLAGS) -o mappedGridExample1 $(mappedGridExample1) $(CLIBS) $(GLIBS) -lm
```

6 Variables contained in a MappedGrid

Here we give a brief overview of some of the most important items that are contained in a MappedGrid.
Define the Ranges $R1, R2, R3$ to define all points on a component grid:

```
const int Start=0, End=1, axis1=0, axis2=1, axis3=2;
CompositeGrid cg;
MappedGrid & mg = cg[grid];
Range R1(mg.dimension(Start,axis1),mg.dimension(End,axis1));
Range R2(mg.dimension(Start,axis2),mg.dimension(End,axis2));
Range R3(mg.dimension(Start,axis3),mg.dimension(End,axis3));
Range ND(0,cg.numberOfDimensions);
```

Recall that we denote the axes of the unit square (or cube) by r_1, r_2, (and r_3). Some arrays such as the `boundaryCondition` array, associate values with each side of a grid. The sides of the grid can be denoted by $r_i = 0$ or $r_i = 1$. These arrays are dimensioned as `boundaryCondition(0:1,0:2)` with

$$boundaryCondition(side, axis) = \text{ value for } r_{axis} = side \quad , \; side = 0,1 \quad , \; axis = 0,1,2 \tag{1}$$

Some arrays, such as the array of vertex coordinates, come in three flavours, `vertex`, `vertex2D` and `vertex1D`. The first is dimensioned `vertex(R1,R2,R3,ND)` and thus looks like an array for a three dimensional grid. When the grid is two-dimensional the Range `R3` will only have 1 point. This array is useful when writing a code that will work in both 3D and 2D. The array `vertex2D(R1,R2,ND)` is only available when the grid is two-dimensional.

- **IntArray boundaryCondition(0:1,0:2)** Boundary condition flags, positive for a real boundary, negative for a periodic boundary and zero for an interpolation boundary.
- **IntArray boundaryDiscretizationWidth(0:2)** Width of the boundary condition stencil.
- **realMappedGridFunction center(R1,R2,R3,ND)** Coordinates of discretization centres.
- **realMappedGridFunction center2D(R1,R2,ND)** Coordinates of discretization centers, for a two-dimensional grid.
- **realMappedGridFunction center1D(R1,ND)** Coordinates of discretization centers, for a one-dimensional grid.
- **realMappedGridFunction centerDerivative(R1,R2,R3,ND,ND)** Derivative of the mapping at the discretization centers.
- **realMappedGridFunction centerDerivative2D(R1,R2,ND,ND)** Derivative at the discretization centers, for a two-dimensional grid.
- **realMappedGridFunction centerDerivative1D(R1,ND,ND)**
- **FloatMappedGridFunction centerJacobian(R1,R2,R3)** Determinant of centerDerivative.
- **IntArray dimension(0:1,0:2)** Dimensions of grid arrays – actual size of the A++ arrays, including ghost-points.
- **IntArray discretizationWidth(0:2)** Interior discretization stencil width (default=3)
- **IntArray gridIndexRange(0:1,0:2)** Index range of gridpoints, excluding ghost points.
- **realArray gridSpacing(0:2)** Grid spacing in the unit square, equal to 1 over the number of grid cells.
- **IntArray indexRange(0:1,0:2)** Index range of computational points, excluding ghostpoints and excluding periodic grid lines on the "End".
- **LogicalR isAllCellCentered** Grid is cell-centred in all directions (variable name misspelled for historial reasons, circa 1776)
- **LogicalR isAllVertexCentered** Grid is vertex-centred in all directions
- **LogicalArray isCellCentered(0:2)** Is this grid cell-centred in each direction.

- **IntArray isPeriodic(0:2)** Grid periodicity, equal one if notPeriodic, derivativePeriodic or functionPeriodic.
- **realMappedGridFunction inverseVertexDerivative(R1,R2,R3,ND,ND)** Inverse derivative of the mapping at the vertices. `inverseVertexDerivative(i1,i2,i3,axis,dir)` is the partial derivative of r_{axis} with respesct to x_{dir}.
- **realMappedGridFunction inverseVertexDerivative2D(R1,R2,ND,ND)** Inverse derivative at the vertices, for a two-dimensional grid.
- **realMappedGridFunction inverseVertexDerivative1D(R1,ND,ND)** Inverse derivative at the vertices, for a one-dimensional grid.
- **realMappedGridFunction inverseCenterDerivative(R1,R2,R3,ND,ND)** Inverse derivative at the discretization centers.
- **realMappedGridFunction inverseCenterDerivative2D(R1,R2,ND,ND)** Inverse derivative at the discretization centers, for a two-dimensional grid.
- **realMappedGridFunction inverseCenterDerivative1D(R1,ND,ND)** Inverse derivative at the discretization centers, for a one-dimensional grid.
- **IntMappedGridFunction mask(R1,R2,R3)** mask array that indicates which points are used and not used.
- **MappingRC mapping** Grid mapping (MappingRC is a reference counted Mapping which behaves like the Mapping class)
- **FloatArray minimumEdgeLength(0:2)** Minimum grid cell-edge length.
- **FloatArray maximumEdgeLength(0:2)** Maximum grid cell-edge length.
- **IntR numberOfDimensions** Number of space dimensions, an `IntR` is basically an `int` (used for reference counting).
- **IntArray numberOfGhostPoints(0:1,0:2)** number of ghost points on each side.
- **realMappedGridFunction vertex(R1,R2,R3,ND)** Vertex coordinates.
- **realMappedGridFunction vertex2D(R1,R2,ND)** Vertex coordinates, for a two-dimensional grid.
- **realMappedGridFunction vertex1D(R1,ND)** Vertex coordinates, for a one-dimensional grid.
- **FloatArray vertexBoundaryNormal[3][2]** Outward normal vectors at the vertices on each boundary. These arrays are dimensioned so that they lie on their respective boundary:
  - vertexBoundaryNormal[0][0](R1.getBase():R1.getBase(),R2,R3,ND),
  - vertexBoundaryNormal[0][1](R1.getBound():R1.getBound(),R2,R3,ND),
  - vertexBoundaryNormal[1][0](R1,R2.getBase():R2.getBase(),R3,ND),
  - vertexBoundaryNormal[1][1](R1,R2.getBound():R2.getBound(),R3,ND),
  - etc.
- **FloatArray centerBoundaryNormal[3][2]** Outward normal vectors at the centers on each boundary.
- **realMappedGridFunction vertexDerivative(R1,R2,R3,ND,ND)** Derivative of the mapping at the vertices, `vertexDeriavtive(i1,i2,i3,axis,dir)` is the partial derivative of x_{axis} with respesct to r_{dir}.
- **realMappedGridFunction vertexDerivative2D(R1,R2,ND,ND)** Derivative of the mapping at the vertices, for a two-dimensional grid.
- **realMappedGridFunction vertexDerivative1D(R1,ND,ND)** Derivative of the mapping at the vertices, for a one-dimensional grid.
- **FloatMappedGridFunction vertexJacobian(R1,R2,R3)** Determinant of vertexDerivative.

One may specify (or change) which arrays are to exist in the `MappedGrid` by calling the `update` function with an integer bit-flag. The values of the bit flag are determined from the following enumerator

```
enum {
  USEmask                    = USEgenericGrid             << 1,
  USEinverseVertexDerivative = USEmask                    << 1,
  USEinverseCenterDerivative = USEinverseVertexDerivative << 1,
  USEvertex                  = USEinverseCenterDerivative << 1,
  USEcenter                  = USEvertex                  << 1,
  USEvertexDerivative        = USEcenter                  << 1,
  USEcenterDerivative        = USEvertexDerivative        << 1,
  USEfaceNormal              = USEcenterDerivative        << 1,
  USEvertexJacobian          = USEfaceNormal              << 1,
  USEcenterJacobian          = USEvertexJacobian          << 1,
  USEvertexBoundaryNormal    = USEcenterJacobian          << 1,
  USEcenterBoundaryNormal    = USEvertexBoundaryNormal    << 1,
  USEmappedGrid              = USEcenterBoundaryNormal // Do not use.
};
```

- **MappedGrid(const String & file, const String & name)** Constructor from database file and name.
- **MappedGrid(Mapping & mapping)** Constructor from a mapping.
- **void updateReferences()** Set references to reference-counted data.
- **void update(const Int what = USEtheUsualSuspects)** Update the grid.

For further details consult the documentation sitting in the chair in Geoff's office.

7 Variables contained in a CompositeGrid

Define the Ranges

```
const int Start=0, End=1, axis1=0, axis2=1, axis3=2;
CompositeGrid cg;
MappedGrid & mg = cg[grid];
Range R1(mg.dimension(Start,axis1),mg.dimension(End,axis1));
Range R2(mg.dimension(Start,axis2),mg.dimension(End,axis2));
Range R3(mg.dimension(Start,axis3),mg.dimension(End,axis3));
Range ND(0,cg.numberOfDimensions());
Range NG(0,cg.numberOfComponentGrids());
Range MG(0,cg.numberOfMultigridLevels());

Range NI(0,cg.numberOfInterpolationPoints(grid));
```

- **IntR numberOfComponentGrids** Number of component grids (MappedGrid's).
- **IntR numberOfDimensions** Number of space dimensions.
- **IntArray numberOfInterpolationPoints(NG)** The number of interpolation points on each component grid.
- **LogicalR interpolationIsAllExplicit**
- **LogicalArray interpolationIsImplicit(NG,NG)**
- **IntArray interpolationWidth(3,NG,NG)** The width of the interpolation stencil (direction, toGrid, fromGrid).
- **realArray interpolationOverlap(3,NG,NG)** The minimum overlap for interpolation (direction, toGrid, fromGrid).
- **ListOfReferenceCountedObjects<realArray> interpolationCoordinates[NG](NI,ND)** Coordinates of interpolation point on component grid "grid" are `interpolationCoordinates[grid](n,axis)` for $0 \le n \le$ `numberOfInterpolationPoints(grid)`.
- **ListOfReferenceCountedObjects<IntArray> interpoleeGrid[NG](NI)** Index of the "interpolee grid", i.e. this is the index of the grid from which we interpolate.
- **ListOfReferenceCountedObjects<IntArray> interpoleeLocation[NG](NI,ND)** Location of interpolation stencil on the interpolee grid, this is the index of the lower left corner of the stencil.
- **ListOfReferenceCountedObjects<IntArray> interpolationPoint[NG](NI,ND)** Indices of interpolation point.
- **ListOfReferenceCountedObjects<realArray> interpolationCondition[NG](NI)** Interpolation condition number.
- **IntGridCollectionFunction mask[NG](R1,R2,R3)** Flag array, positive for discretization point, negative for interpolation point, zero for unused point.
- **ListOfReferenceCountedObjects<MappedGrid> grid[NG]** Here is the list of MappedGrid's.

Here are variables related to multigrid levels

- **IntR numberOfMultigridLevels**
- **IntArray coarseToFineWidth(0:2,NG,MG)** Prolongation stencil width
- **IntArray coarseToFineIsImplicit(NG,MG)** Prolongation is always implicit.
- **IntArray fineToCoarseWidth(0:2,NG,MG)** Restriction stencil width
- **IntArray fineToCoarseIsImplicit(NG,MG)** Restriction is always implicit.
- **IntArray fineToCoarseFactor(0:2,NG,MG)** Ratio of this to coarser level
- **ListOfReferenceCountedObjects<CompositeGrid> compositeGrid**

References

[1] D. L. Brown, *Overture operator classes for finite volume computations on overlapping grids, user guide*, Tech. Rep. UCRL-MA-133649, Lawrence Livermore National Laboratory, 1998.

[2] D. L. Brown and W. D. Henshaw, *Adaptive mesh refinement routines for Overture*, Research Report UCRL-MA-140918, Lawrence Livermore National Laboratory, 2000.

[3] W. D. Henshaw, *A fourth-order accurate method for the incompressible Navier-Stokes equations on overlapping grids*, J. Comput. Phys., 113 (1994), pp. 13–25.

[4] ——, *Finite difference operators and boundary conditions for Overture, user guide*, Research Report UCRL-MA-132231, Lawrence Livermore National Laboratory, 1998.

[5] ——, *Grid functions for Overture, user guide*, Research Report UCRL-MA-132231, Lawrence Livermore National Laboratory, 1998.

[6] ——, *Mappings for Overture, a description of the Mapping class and documentation for many useful Mappings*, Research Report UCRL-MA-132239, Lawrence Livermore National Laboratory, 1998.

[7] ——, *Ogen: An overlapping grid generator for Overture*, Research Report UCRL-MA-132237, Lawrence Livermore National Laboratory, 1998.

[8] ——, *Oges user guide, a solver for steady state boundary value problems on overlapping grids*, Research Report UCRL-MA-132234, Lawrence Livermore National Laboratory, 1998.

[9] ——, *Ogshow: Overlapping grid show file class, saving solutions to be displayed with plotStuff, user guide*, Research Report UCRL-MA-132235, Lawrence Livermore National Laboratory, 1998.

[10] ——, *Plotstuff: A class for plotting stuff from Overture*, Research Report UCRL-MA-132238, Lawrence Livermore National Laboratory, 1998.

[11] ——, *A primer for writing PDE codes with Overture*, Research Report UCRL-MA-132231, Lawrence Livermore National Laboratory, 1998.

[12] ——, *Other stuff for Overture, user guide, version* 1.0, Research Report UCRL-MA-134292, Lawrence Livermore National Laboratory, 1999.

[13] ——, *OverBlown: A fluid flow solver for overlapping grids, reference guide*, Research Report UCRL-MA-134289, Lawrence Livermore National Laboratory, 1999.

[14] ——, *OverBlown: A fluid flow solver for overlapping grids, user guide*, Research Report UCRL-MA-134288, Lawrence Livermore National Laboratory, 1999.

[15] ——, *Time step determination for pdes with applications to programs written with Overture*, Research Report UCRL-MA-134300, Lawrence Livermore National Laboratory, 1999.

[16] W. D. Henshaw, H.-O. Kreiss, and L. G. M. Reyna, *A fourth-order accurate difference approximation for the incompressible Navier-Stokes equations*, Comput. Fluids, 23 (1994), pp. 575–593.

[17] D. Quinlan, *A++/P++ class libraries*, Research Report LA-UR-95-3273, Los Alamos National Laboratory, 1995.

Index