LA-UR 96-3894

TITLE: **A PRIMER FOR WRITING OVERLAPPING GRID CODES IN C++**

RECEIVED
JAN 21 1997
OSTI

AUTHOR(S): William D. Henshaw

SUBMITTED TO: For Electronic Distribution on the Web

Los Alamos

Los Alamos National Laboratory
Los Alamos New Mexico 87545

DISCLAIMER

A Primer for Writing Overlapping Grid Codes in C++

William D. Henshaw
Computing, Information and Communications Division
Los Alamos National Laboratory
Los Alamos, NM, 87545 [1]

October 16, 1996

Abstract: We describe how to write C++ programs to solve partial differential equations on overlapping grids. We use the grid contruction program Ogen to create overlapping grids. We use the parallel array class library A++ to write efficient and portable serial or parallel code.

Contents

[1]This work was partially supported by grant N00014-93-C-0200 from the Office of Naval Research

1 Introduction

This is a primer for writing C++ codes to solve partial differential equations (PDEs) on overlapping grids. The reader is assumed to be familiar with the A++ (parallel) array class library [?]. The reader is also assumed to be at least slightly familiar with the overlapping grid generator Ogen and the notion of an overlapping or composite grid.

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

- **MappedGrid** : A logically rectangular grid that is defined by a mapping from the unit line (square or cube) into cartesian space. A MappedGrid contains a mapping function as well as information such as boundary conditions, periodicity, grid point coordinates etc.
- **CompositeGrid** : An "overlapping composite grid". Each component grid of a CompositeGrid is a MappedGrid. The grid generator Ogen, for example, can generate a CompositeGrid.
- **realMappedGridFunction** : A grid function that holds a solution on a MappedGrid; this is a slightly glorified A++ array.
- **realCompositeGridFunction** : A grid function that holds a solution (such as the pressure or velocity) on a CompositeGrid.
- **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.
- **MappedGridOperators, CompositeGridOperators** : classes used with grid functions to define spatial derivatives and to apply boundary conditions.
- **Oges** : The overlapping grid equation solver class that can be used to solve systems of boundary value problems such as Poisson's equation.
- **Cgsh** : 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.

Other documents of interest are

- A++ Quick Reference Card : `/n/c3servex/dquinlan/A++P++/DOCS/Quick_Reference_Card.tex`
- Grid and grid function documentation : `/n/c3servet/henshaw/res/gf/gf.tex`
- Finite difference operators and boundary conditions : `/n/c3servet/henshaw/res/gf/op.tex`
- Mapping class documentation : `/n/c3servet/henshaw/res/mapping/mapping.tex`
- Show file documentation : `/n/c3servet/henshaw/res/ogshow/ogshow.tex`
- Interactive plotting : `/n/c3servet/henshaw/res/ogshow/PlotStuff.tex`
- Oges "Equation Solver" documentation : `/n/c3servet/henshaw/res/oges/oges.tex`
- Interactive grid generation documentation : `/n/c3servet/henshaw/res/cgsh/ogen.tex`

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

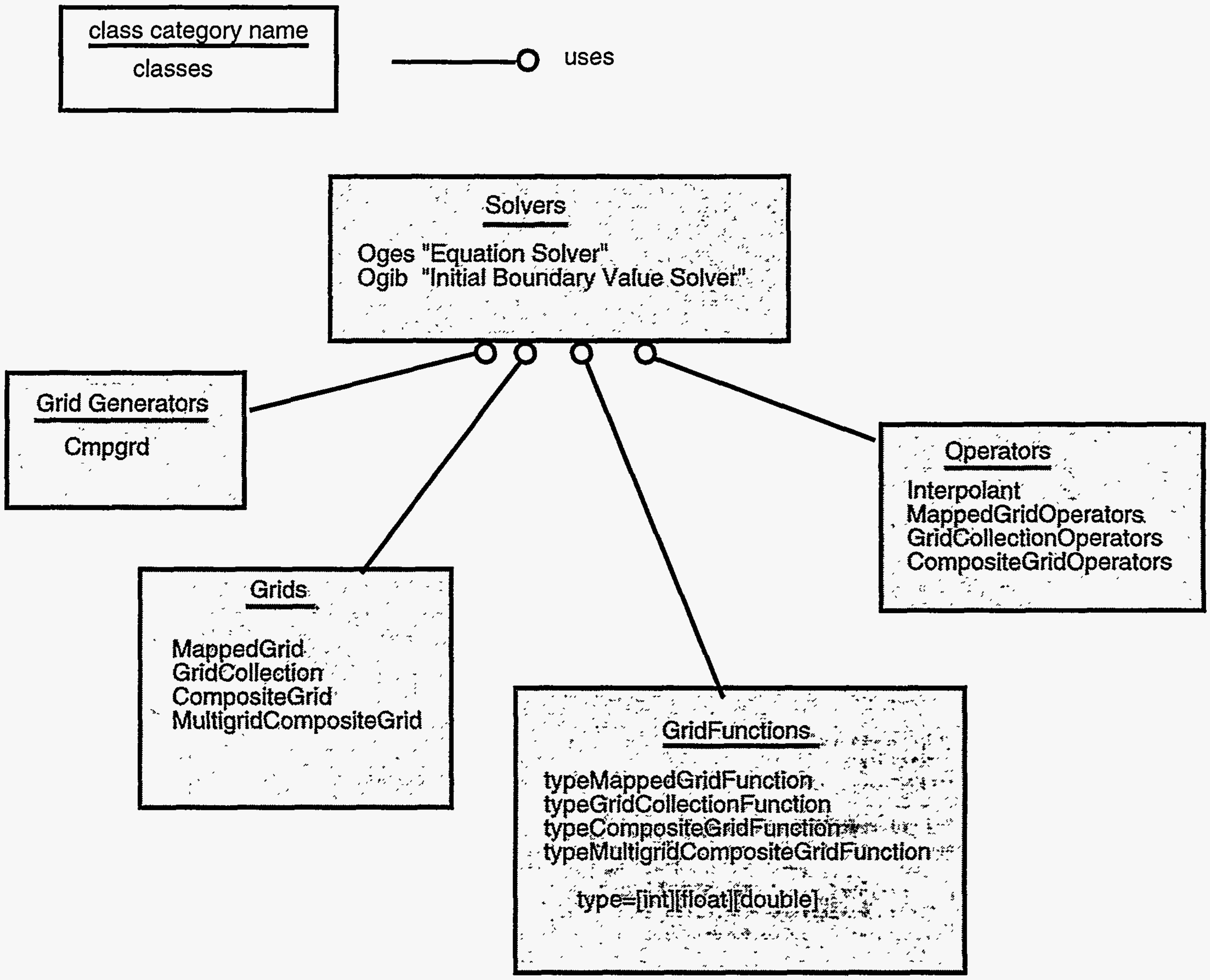

Figure 1: An overview of the Overture classes

2 Getting Started with MappedGrid's

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

(file /n/c3servet/henshaw/res/primer/mappedGridExample1.C)

```
//================================================================================
//  Overture, MappedGrid Example 1
//    o demonstrate the use of a Mapping, MappedGrid, MappedGridFunction
//      and MappedGridOperators
//    o use PlotStuff to display the results
//
//  Bill Henshaw
//================================================================================
#include "Overture.h"
#include "PlotStuff.h"
#include "Square.h"
#include "MappedGridOperators.h"

int
main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  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.update();                                          // create default variables

  Range all;
  realMappedGridFunction u(mg,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;

  // mg.dimension(2,3) : 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

  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;

  PlotStuff ps;                                 // create a PlotStuff object
  PlotStuffParameters psp;                      // This object is used to change plotting parameters

  String answer;
  String menu[] = { "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
      ps.contour(u,psp);                        // contour/surface plots
    }
    else if( answer=="grid" )
    {
      ps.plot(mg);                              // plot the composite grid
    }
    else if( answer=="stream lines" )
    {
      ps.streamLines(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;
    }
  }

  return 0;
}
```

2.2 mappedGridExample2: Mapping's, MappedGrid's, MappedGridFunction's

(file /n/c3servet/henshaw/res/primer/mappedGridExample2.C)

```
//===============================================================================
//  Overture example 2
//     o solve a convection-diffusion equation on an annulus.
//     o plot intermediate results,
//     o use operators to apply boundary conditions
//
//  Bill Henshaw
//===============================================================================
#include "Overture.h"
#include "PlotStuff.h"
#include "Annulus.h"
#include "MappedGridOperators.h"

int
main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  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;
  // mg.dimension(2,3) : 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;                                   // 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 )
    {
      sprintf(buffer,"Solution at time t=%e",t);
      psp.set(GI_TOP_LABEL,buffer);  // set title
      ps.contour( u,psp );
    }

    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) );
    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();
  }

  return 0;
}
```

2.3 mappedGridExample3: Mapping's, MappedGrid's, MappedGridFunction's

(file /n/c3servet/henshaw/res/primer/mappedGridExample3.C)

```
//==============================================================================
//  Overture example 3
//     o solve a convection-diffusion equation on an annulus.
//     o demonstrate the NameList class for inputing parameters
//     o demontrate updateToMatchGrid, applying BC's explicitly
//
//  Bill Henshaw
//==============================================================================
#include "Overture.h"
#include "PlotStuff.h"
#include "Annulus.h"
#include "MappedGridOperators.h"
#include "NameList.h"

int
main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  // Set default values for parameters. These can be optionally changed below
  int numberOfTimeSteps=100;
  real dt=.005;
  IntegerArray bc(2,3); bc=1;
  // The NameList object allows one to read in values by name
  NameList nl;
  String 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;
    cin >> 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== "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;
```

```
  // mg.dimension(2,3) : 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;                                   // 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;
  for( int step=0; step<numberOfTimeSteps; step++ )
  {
    if( step % 10 == 0 )
    {
      sprintf(buffer,"Solution at time t=%e",t);
      psp.set(GI_TOP_LABEL,buffer);  // set title
      ps.contour( u,psp );
    }
    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) );
    t+=dt;
    // 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)=0.;
        }
      }
    u.periodicUpdate();  // swap periodic edges
  }

  return 0;
}
```

2.4 mappedGridExample4: Mapping's, MappedGrid's, MappedGridFunction's

(file /n/c3servet/henshaw/res/primer/mappedGridExample4.C)

```
//===========================================================================
//  Overture example:
//     o test a mapping
//     o interactively change a mapping
//     o save the mapping to a data-base file
//     o read the mapping from a data-base file
//
//  Bill Henshaw
//===========================================================================
#include "Overture.h"
#include "PlotStuff.h"
#include "ChannelMapping.h"
#include "MappingInformation.h"
#include "HDF_DataBase.h"

int
main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  ChannelMapping channel;

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

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


  // Make interactive changes to the mapping
  PlotStuff ps;                          // 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),r(1),x(0),x(1),
         xr(0,0),xr(1,0),xr(0,1),xr(1,1));

  return 0;
}
```

2.5 mappedGridExample5: Mapping's, MappedGrid's, MappedGridFunction's

(file /n/c3servet/henshaw/res/primer/mappedGridExample5.C)

```
//==============================================================================
//
// Overture example:
//   o  Use OGPolyFunction and OGTrigFunctions classes to generate true solutions
//   o  optionally turn plotting on or off
//
// Bill Henshaw
//=============================================================================
#include "Overture.h"
#include "PlotStuff.h"
#include "Square.h"
#include "Annulus.h"
#include "MappedGridOperators.h"
#include "NameList.h"
#include "OGFunction.h"
#include "OGTrigFunction.h"
#include "OGPolyFunction.h"


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


int
main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking


  // 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;
  String 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;
    cin >> 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 *exact;
if( forcingOption==poly )
{
  int degreeOfSpacePolynomial = 2;
  int degreeOfTimePolynomial = 1;
  int nComp = 1;
  exact = 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)
  exact = new OGTrigFunction(fx, fy, fz, ft);
}
else if( forcingOption!=0 )
{
  cout << "Unknown forcing option = " << forcingOption << endl;
  forcingOption=noForcing;
}

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->u(mg,I1,I2,I3,0,0.);
else
  u=1.;
```

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

  PlotStuff ps(plotOption);        // 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
      ps.contour( u,psp );
    }

    u+=dt*( (-a)*u.x()+(-b)*u.y()+nu*(u.xx()+u.yy()) );
    if( forcingOption > 0 )
    {
      u(I1,I2,I3)+=dt*(exact->ut(mg,I1,I2,I3,0,t)
             + a*exact->ux(mg,I1,I2,I3,0,t) + b*exact->uy(mg,I1,I2,I3,0,t)
             - nu*( exact->uxx(mg,I1,I2,I3,0,t) + exact->uyy(mg,I1,I2,I3,0,t) ) );
    }
    t+=dt;
    // 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
          if( forcingOption > 0 )
          {
            getBoundaryIndex(mg.gridIndexRange,side,axis,Ib1,Ib2,Ib3);
            u(Ib1,Ib2,Ib3)=exact->u(mg,Ib1,Ib2,Ib3,0,t);
          }
          else
          {
            u(Ib1,Ib2,Ib3)=0.;
          }
        }
      }
    u.periodicUpdate();  // swap periodic edges

    real error = max(abs( u(I1,I2,I3)-exact->u(mg,I1,I2,I3,0,t)));
    cout << "t=" << t << ", error =" << error << endl;
  }

  return 0;
}
```

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 /n/c3servet/henshaw/res/primer/example1.C)

```
#include "Overture.h"

int
main()
{
  String 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
  }

  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 4 and 5 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 /n/c3servet/henshaw/res/cgsh/square5.hdf.

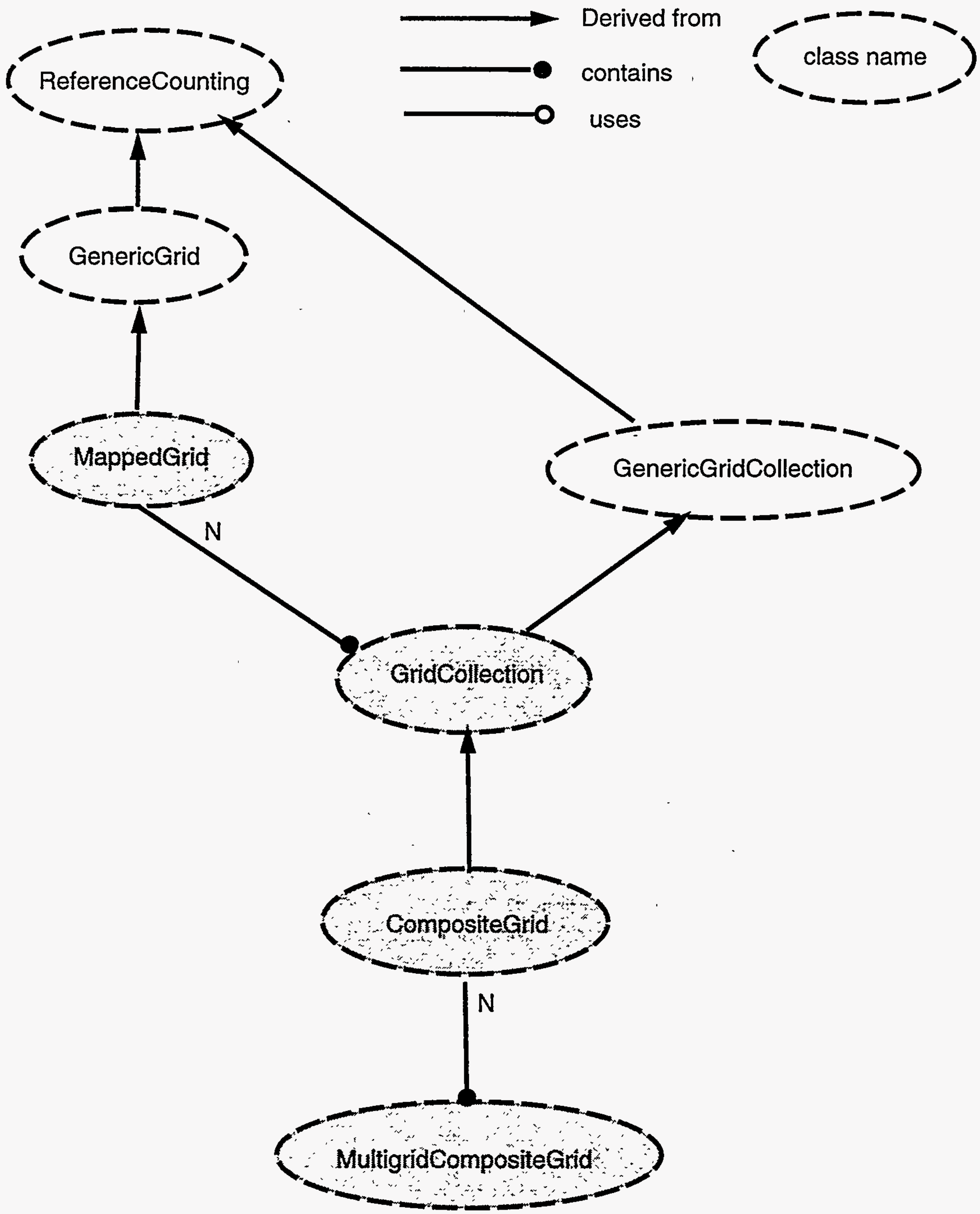

Figure 2: Class diagram for grid classes

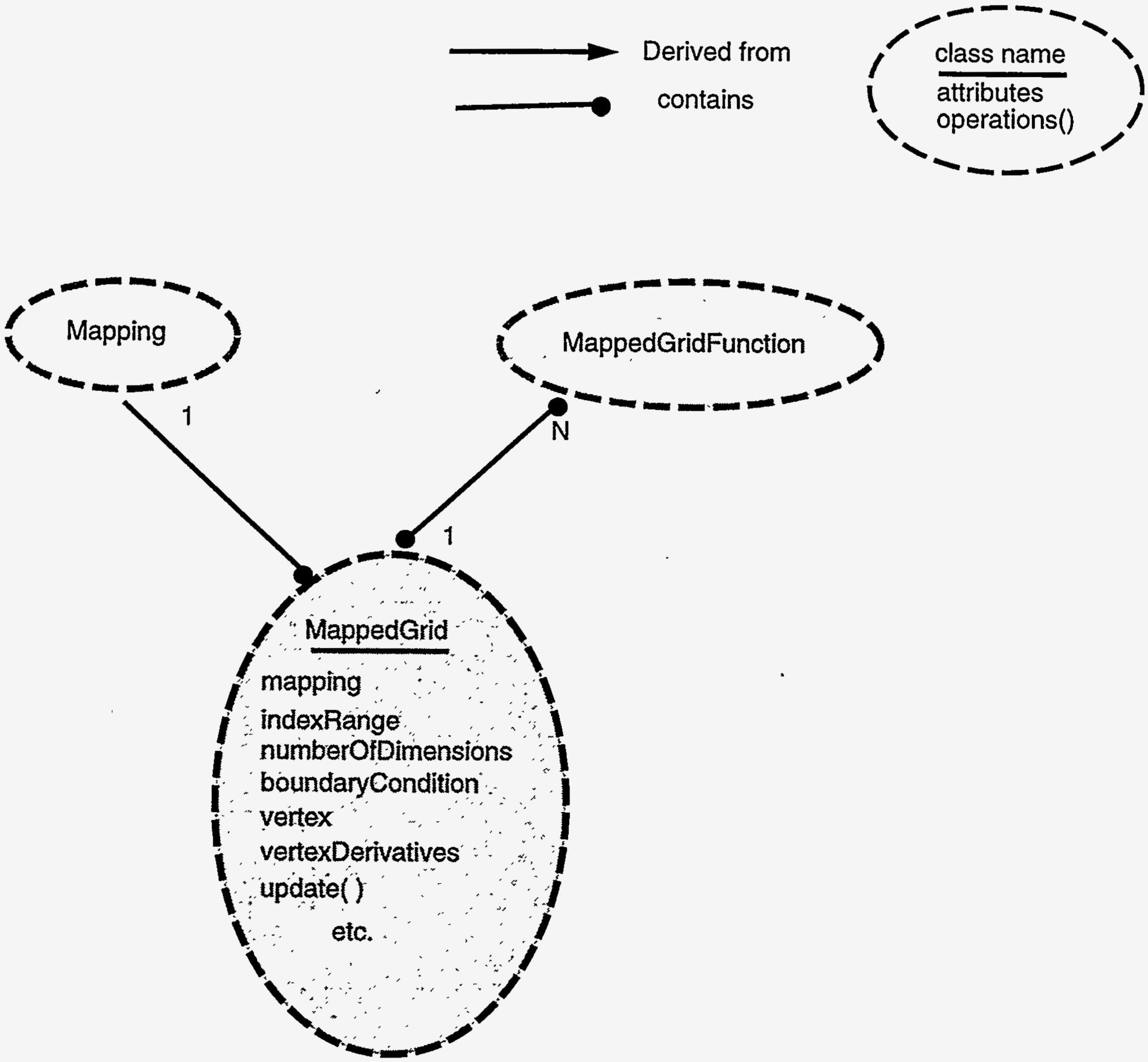

Figure 3: Class diagram for a MappedGrid

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 [?]. A `realCompositeGridFunction` contains a list of "realMappedGridFunctions", one `realMappedGridFunction` for each component grid.

(file `/n/c3servet/henshaw/res/primer/example2.C`)

```
#include "Overture.h"

int main()
{
  const int Start=0, End=1, axis1=0, axis2=1, axis3=2;

  String 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)");

  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.

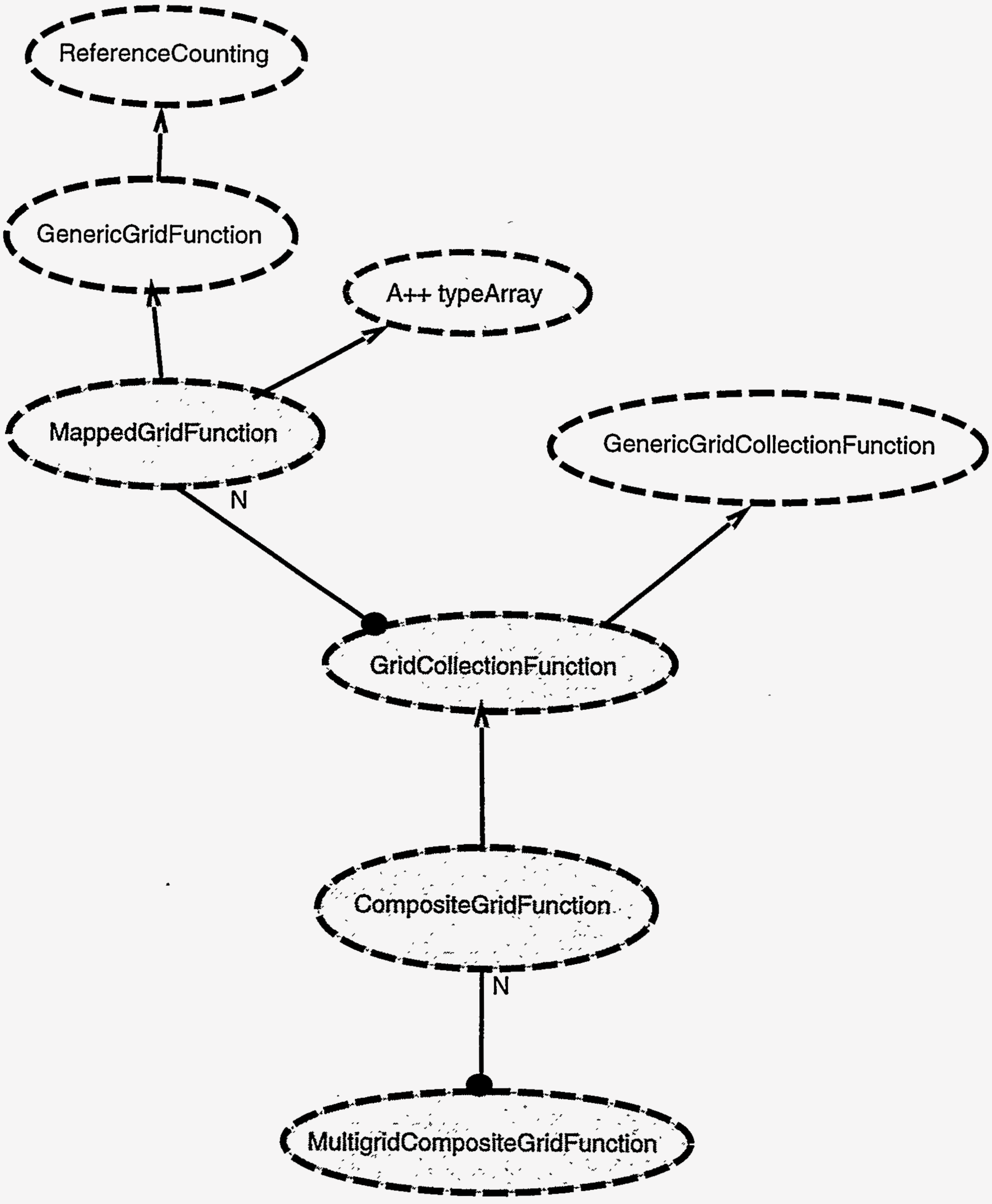

Figure 4: Class diagram for grid function classes

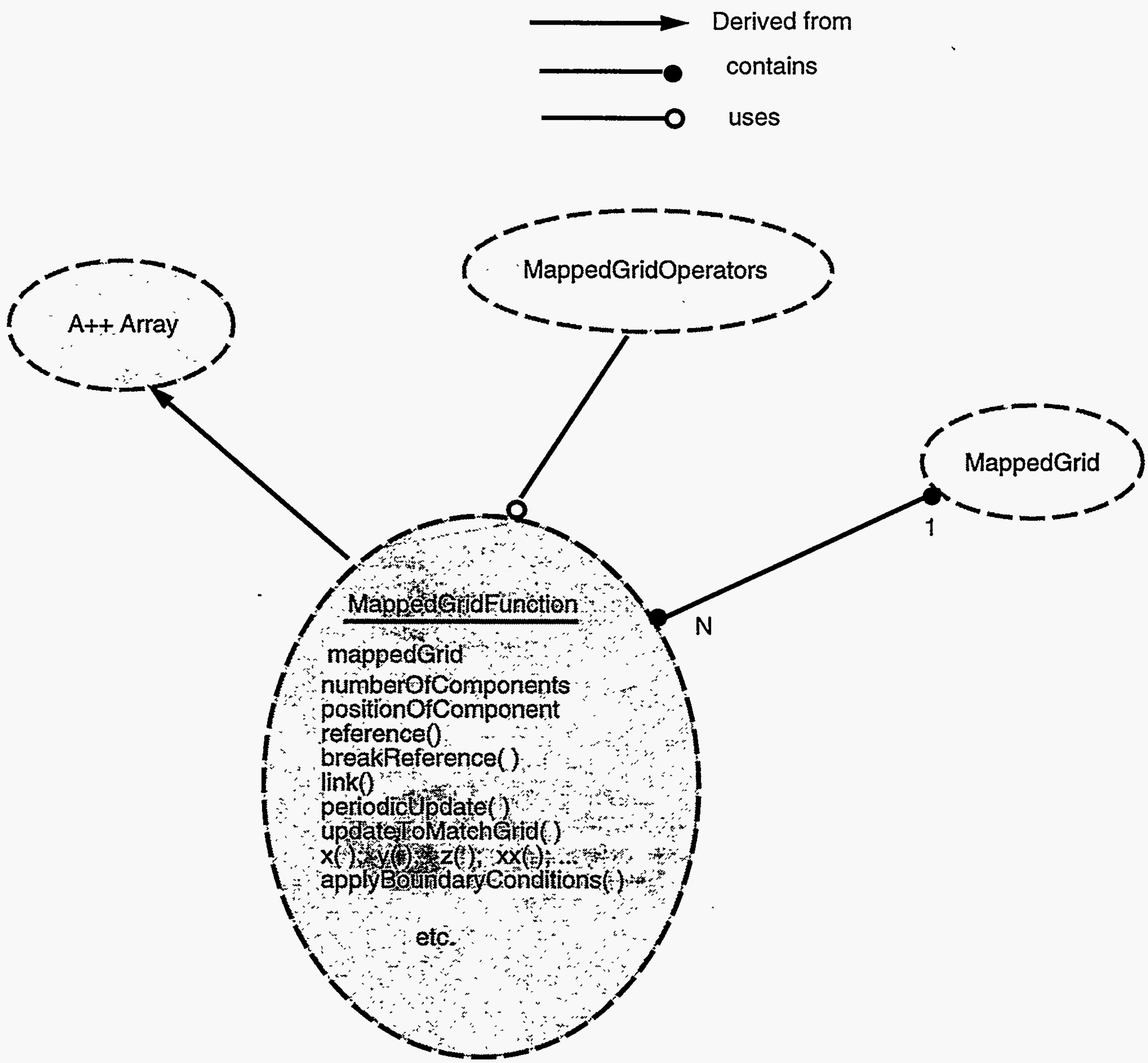

Figure 5: Class diagram for a MappedGridFunction

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 /n/c3servet/henshaw/res/primer/example3.C)

```
#include "Overture.h"

int main()
{
  const int Start=0, End=1, axis1=0, axis2=1, axis3=2;

  String 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.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");

  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. (Note how the mask array is accessed – it is an "intCompositeGridFunction" in the CompositeGrid; it does not exist in a MappedGrid.) 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 /n/c3servet/henshaw/res/primer/example4.C)

```
//===============================================================================
//  Test the Overlapping Grid Show file class Ogshow
//===============================================================================
#include "Overture.h"
#include "Ogshow.h"

int main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  String 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
  }

  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 the c-shell script `/n/c3servet/henshaw/res/primer/show.s` to run plotStuff to display the results from this example.

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 /n/c3servet/henshaw/res/primer/example5.C)

```
//===========================================================================
//  Differentiating Grid Functions
//===========================================================================
#include "Overture.h"
#include "CompositeGridOperators.h"

int main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  String 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

  realCompositeGridFunction u(cg),ux(cg);                    // create two composite grid functions

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

  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)");

  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

    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;
  }
  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 /n/c3servet/henshaw/res/primer/example6.C)

```
//===========================================================================
//  Test the Differentiable Grid Functions
//    o Solve: u.t + a*u.x + b*u.y = viscosity*( u.xx + u.yy )
//    o Save the solutions in a show file
//===========================================================================
#include "Overture.h"
#include "Ogshow.h"
#include "CompositeGridOperators.h"

int main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

  String nameOfOGFile, nameOfShowFile;
  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;

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

  Interpolant interpolant(cg);                                // Make an interpolant

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

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

  Range all;
  realCompositeGridFunction u(cg,all,all,all,1);              // create a grid function
  u.setOperators(operators);
  u.setName("u");                                             // name the grid function
  // u.setName("u",0);                                           // name the component

  u=1.;                                                       // initial condition
  real t=0, dt=.01;                                           // initialize time and time step
  real a=1., b=1., viscosity=.1;                              // initialize parameters

  char buffer[80];                                            // buffer for sprintf
  int numberOfTimeSteps=5;
  for( int i=1; i<=numberOfTimeSteps; i++ )                   // take some time 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.finishBoundaryConditions();
  }

  return 0;

}
```

Use the c-shell script `/n/c3servet/henshaw/res/primer/show.s` to run plotStuff to display the results from this example.

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.

Currently, computing derivatives in this way will not be so efficient. An efficient way to compute derivatives is described 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.

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 /n/c3servet/henshaw/res/primer/example7.C)

```
//===============================================================================
//  Primer: Example 7 : Using Oges to solve Poisson's equation
//===============================================================================
#include "Overture.h"
#include "CompositeGridOperators.h"
#include "Oges.h"

main()
{
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking
  String 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();
  cout << "example7: number of dimensions =" << cg.numberOfDimensions
       << ", or =" << cg[0].numberOfDimensions << endl;

  // make a grid function to hold the coefficients
  Range all;
  int stencilSize=pow(3,cg.numberOfDimensions)+1;  // 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.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++ )
  {
    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

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

  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 /n/c3servet/henshaw/res/ogshow/PlotStuff.tex.

Here is an example code that uses the `PlotStuff` class to plot various objects from the Overture class (file `/n/c3servet/henshaw/res/primer/example8.C`)

```
//===============================================================================
//  Primer: Example 8: Interactive plotting with PlotStuff
//==============================================================================
#include "Overture.h"
#include "PlotStuff.h"

int
main()
{
  ios::sync_with_stdio();      // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);   //  Turn on A++ array bounds checking
  String 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));
  }

  PlotStuff ps;                                  // create a PlotStuff object
  PlotStuffParameters psp;                       // This object is used to change plotting parameters

  String answer;
  String menu[] = { "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
      ps.contour(u,psp);                         // contour/surface plots
    }
    else if( answer=="grid" )
    {
      ps.plot(cg);                               // plot the composite grid
    }
    else if( answer=="stream lines" )
    {
      ps.streamLines(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;
    }
  }

  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: 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.

(file `/n/c3servet/henshaw/res/primer/move1.C`)

```
#include "Cgsh.h"
#include "PlotStuff.h"
#include "MatrixTransform.h"

//
// Moving Grid Example: read in a grid from a data-base file, rotate a component grid
// and recompute the overlapping grid
//
int
main()
{
  Mapping::debug=7;
  ios::sync_with_stdio();     // Synchronize C++ and C I/O subsystems
  Index::setBoundsCheck(on);  //  Turn on A++ array bounds checking

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

  // 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[0].update();
  cg[1]=cg[0];                                      // copy cg[0] into cg[1]

  // Rotate component grid 1 (do this by changing the mapping)
  int gridToMove=1;
  Mapping & mappingToMove = *(cg[0][gridToMove].mapping.mapPointer);

  // 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[1][gridToMove].reference(transform1);

  // 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(2);
  hasMoved    = LogicalFalse;
  hasMoved(gridToMove) = LogicalTrue;  // Only this grid will move.
  char buff[80];

  PlotStuff ps;                                     // for plotting
  // Here is the overlapping grid generator
  Cgsh gridGenerator(ps);

  int numberOfSteps=5;
  real deltaAngle=5.*Pi/180.;
  // ---- Move the grid a bunch of times.----
  for (int i=1; i<=numberOfSteps; i++)
  {
    int newCG = i % 2;        // new grid
    int oldCG = (i+1) % 2;    // old grid

    ps.plot(cg[oldCG]);      // plot the current overlapping grid

```

```
    // Rotate the grid by rotating the mapping
    // After the first step we must double the angle since we start from the old grid
    real angle = i==1 ? deltaAngle : deltaAngle*2.;
    if( newCG==0 )
      transform0.rotate(axis3,angle);
    else
      transform1.rotate(axis3,angle);

    // Update the overlapping newCG, starting with and sharing data with oldCG.
    gridGenerator.updateOverlap(cg[newCG], cg[oldCG], hasMoved);

  }
  return 0;
}
```

3.10 Makefile

Here is the Makefile I used for the examples in this primer.
(file /n/c3servet/henshaw/res/primer/Makefile)

```
#
#  This Makefile file assumes that the following environmental variables are set from my .cshrc
#
#    APLusPlus    - gives the location of the version of A++ I use
#    OpenGL       - gives the location of the Mesa OpenGL libraries
#    HDF          - gives the location of the HDF database libraries
#    Overture     - give the location of Overture
#    MOTIF        - gives the location for Motif
#    XLIBS        - gives the location for X
#    GL_LIBS      - a list of OpenGL libraries
#       example:    setenv GL_LIBS  "-lGLw -lGL -lGLU"
#                   setenv GL_LIBS  "-lMesaaux -lMesatk -lMesaGLU -lMesaGL"
#    FORTRAN_LIBS - a list of fortran libraries
#       example: setenv  FORTRAN_LIBS -lF77 -lM77 -lV77 -lnsl
#

# Overture=         /n/c19s3/Overture/Overture.v2.g
# Overture=         /n/c3servet/henshaw/Overture.g
OvertureInclude=  $(Overture)/include
OvertureLib=      $(Overture)/lib


A++ =.            $(APlusPlus)
OpenGLInclude = $(OpenGL)/include

Include =     -I. -I$(A++)/include -I$(OvertureInclude) -I$(OpenGLInclude)
CC=         CC
CCFLAGS=    -g -cg92 $(Include)
# dp: CCFLAGS=    -g -cg92 -DDOUBLE $(Include)
FC=         f77
FFLAGS=
# dp: FFLAGS = -r8 -i4
CLIBS=   -L$(OvertureLib) -lOverture -lOverture_static -L$(A++) -lA++ -lA++_static \
         -L$(HDF)/lib -lmfhdf -ldf -ljpeg -lz

FLIBS=   $(FORTRAN_LIBS)
GLIBS=   -L$(OpenGL)/lib $(GL_LIBS)  \
         -L$(MOTIF)/lib -lXm -L$(XLIBS) -lXt -lXmu -lXi -lXext -lX11 -lm

# These are for purify, uncomment the lines below (#PURIFY ...) to use purify
PFLAGS  = -first-only=yes -leaks-at-exit=yes -inuse-at-exit=yes -always-use-cache-dir=yes
PDIR    = `purify -print-home-dir`
PURIFY  =
# PURIFY  = purify $(PFLAGS)
# PSTUBS  = $(PDIR)/purify_stubs.a
PSTUBS  =
# PURIFY_OPTIONS = -I$(PDIR)

all = example1 example2 example3 example4 example5 example6 example7 example8
all: $(all);

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

primer:
        latex primer

doc:
        latex primer
        latex primerLong
        dvips -f primerLong.dvi > primer.ps
        gzip -9f primer.ps
        latex gettingOverture
        dvips -f gettingOverture.dvi > gettingOverture.ps

```

```
mappedGridExample1 = mappedGridExample1.o
mappedGridExample1: $(mappedGridExample1)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o mappedGridExample1 $(mappedGridExample1) \
	$(CLIBS) $(FLIBS)  $(GLIBS)

mappedGridExample2 = mappedGridExample2.o
mappedGridExample2: $(mappedGridExample2)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o mappedGridExample2 $(mappedGridExample2) \
	$(CLIBS) $(FLIBS)  $(GLIBS)

mappedGridExample3 = mappedGridExample3.o
mappedGridExample3: $(mappedGridExample3)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o mappedGridExample3 $(mappedGridExample3) \
	$(CLIBS) $(FLIBS) $(GLIBS)

mappedGridExample4 = mappedGridExample4.o ChannelMapping.o
mappedGridExample4: $(mappedGridExample4)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o mappedGridExample4 $(mappedGridExample4) \
	$(CLIBS) $(FLIBS) $(GLIBS)

mappedGridExample5 = mappedGridExample5.o
mappedGridExample5: $(mappedGridExample5)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o mappedGridExample5 $(mappedGridExample5) \
	$(CLIBS) $(FLIBS) $(GLIBS)


example1 = example1.o
example1: $(example1)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example1 $(example1) $(CLIBS) $(FLIBS)  $(GLIBS)

example2 = example2.o
example2: $(example2)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example2 $(example2) $(CLIBS) $(FLIBS)  $(GLIBS)

example3 = example3.o
example3: $(example3)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example3 $(example3) $(CLIBS) $(FLIBS) $(GLIBS)

example4 = example4.o
example4: $(example4)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example4 $(example4) $(CLIBS) $(FLIBS) $(GLIBS)

example5 = example5.o
example5: $(example5)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example5 $(example5) $(CLIBS) $(FLIBS) $(GLIBS)

example6 = example6.o
example6: $(example6)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example6 $(example6) $(CLIBS) $(FLIBS)  $(GLIBS)

example7 = example7.o
example7: $(example7)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example7 $(example7) $(CLIBS) $(FLIBS) $(GLIBS)

example8 = example8.o
example8: $(example8)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o example8 $(example8) $(CLIBS) $(FLIBS) $(GLIBS)

move1 = move1.o
move1: $(move1)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o move1 $(move1) $(CLIBS) $(FLIBS) $(GLIBS)

tcm3 = tcm3.o
tcm3 = tcm3.o
tcm3: $(tcm3)
	$(PURIFY) $(CC) $(CCFLAGS) $(PURIFY_OPTIONS) -o tcm3 $(tcm3) $(CLIBS) $(FLIBS) -lftn $(GLIBS)

clean:
	rm *.o mappedGridExample1 mappedGridExample2 mappedGridExample3 mappedGridExample4 mappedGridExample5 \
	   example1 example2 example3 example4 example5 example6 example7 example8

.PRECIOUS:
```

4 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

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

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.

5 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 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 $0 \leq n \leq$ `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.