Overture: Object-Oriented Tools for Overset Grid Applications

D.L. Brown
W.D. Henshaw
D.J. Quinlan

This paper was prepared for submittal to the
17th American Institute of Aeronautics and Astronautics
Applied Aerodynamics Conference
Norfolk, VA
June 28-July 1, 1999
DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.
OVERTURE: OBJECT-ORIENTED TOOLS FOR OVERSET GRID APPLICATIONS

David L. Brown, William D. Henshaw and Daniel J. Quinlan
Centre for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
Livermore, California, 94551.

ABSTRACT

The Overture framework is an object-oriented environment for solving partial differential equations in two and three space dimensions. It is a collection of C++ libraries that enables the use of finite difference and finite volume methods at a level that hides the details of the associated data structures. Overture can be used to solve problems in complicated, moving geometries using the method of overlapping grids. It has support for grid generation, difference operators, boundary conditions, data-base access and graphics. Short sample code segments are presented to show the power of this approach.

INTRODUCTION

The Overture framework is a collection of C++ libraries that provide tools for solving partial differential equations. Overture can be used to solve problems in complicated, moving geometries using the method of overlapping grids (also known as overset or Chimera grids). Overture includes support for geometry, grid generation, difference operators, boundary conditions, data-base access and graphics.

An overlapping grid consists of a set of logically rectangular grids that cover a domain and overlap where they meet. This method has been used successfully over the last decade and a half, primarily to solve problems involving fluid flow in complex, often dynamically moving, geometries. Solution values at the overlap are determined by interpolation. The overlapping grid approach is particularly efficient for rapidly generating high-quality grids for moving geometries. As the component grids move only the boundary points to be interpolated change, the grid points do not have to be regenerated. The component grids are structured so that efficient and fast finite-difference algorithms can be utilized. We use adaptive mesh refinement to resolve fine features of the solution. The design of Overture has evolved over the past 15 years or so from the Fortran based CMPGRD environment to the current C++ version. Although the Fortran implementation was used for complicated three-dimensional adaptive and moving grid computations, the programs were difficult to write and maintain. Overture was designed to have at least all the functionality of the Fortran code but to be as easy as possible to use; indeed, an entire PDE solver on an overlapping grid can be written on a single page (see section ).

Overture is an object-oriented framework. In the past a typical Fortran code would use a procedural model where subroutines and functions are the fundamental building blocks and data is passed to and from these procedures. In Overture the fundamental building blocks are objects such as grids and grid functions. These objects can be manipulated at a high level. Details of the implementation, such as how a grid is stored, are hidden from the user. In the object-oriented world this is known as data encapsulation. One major benefit of encapsulation is that changes can be made to the implementation of an object without forcing changes to be made to the code that uses the object. An object such as a grid function contains not only data (such as the values of density at each point on a grid) but also implements functions that operate on the data (these functions are often called methods in object-oriented terminology). Thus a grid function (which may live on a collection of grids on a overlapping grid) will know what it means to add itself to another grid function, which would be expressed in a C++ code as \[ u = u + w. \] The '+' operator and the '=' operator are defined by
the grid function class, a process known as operator overloading.

A common complaint of object-oriented languages such as C++ is that it is easy to write code that is elegant but runs many times slower than fortran. It is certainly true that since C++ is a much richer language than fortran that it is easy to write inefficient code. However, it is also true that by writing a code in C++ with basically a fortran style that it is possible to achieve the exact same performance of fortran (while still benefiting from some of the nice features of C++). Overture has been designed to be used at different levels to allow users to obtain full performance at the cost of writing at a lower level. Thus one can either operate at a high level to write an entire code (with a decrease in performance) or one may only use C++ to manage the complex data structures while calling Fortran or C routines to perform computationally intensive tasks. In the future it is likely that even the high level code can be made to run as fast as the low level approach. We are currently working on a preprocessor that will automatically convert high level C++ code into efficient C code. Initial results show that full fortran performance can be obtained using the preprocessor.

There are a number of other very interesting projects developing scientific object-oriented frameworks. These include the SAMRAI framework for structured adaptive mesh refinement 22, PETSc (the Portable Extensible Toolkit for Scientific Computation) 18, POOMA (Parallel Object Oriented Methods and Applications) 19 and Diffpack 10.

THE OVERTURE FRAMEWORK

The main class categories that make up Overture are as follows:

- **Arrays**: describe multidimensional arrays using A[][]/P[]. A[][] provides the serial ar-
ray objects, and P++ provides the distribution and interpretation of communication required for their data parallel execution.

- **Mappings 15**: define transformations such as curves, surfaces, areas, and volumes. These are used to represent the geometry of the computational domain.

- **Grids 9, 14**: define a discrete representation of a mapping or mappings. These include single grids, and collections of grids; in particular composite overlapping grids.

- **Grid functions 14**: storage of solution values, such as density, velocity, pressure, defined at each point on the grid(s).

- **Operators 4, 13**: provide discrete representations of differential operators and boundary conditions.

- **Grid generation 16**: the Ogen overlapping grid generator automatically constructs an overlapping grid given the component grids.

- **Plotting 17**: a high-level interface based on OpenGL allows for plotting Overture objects.

- **Adaptive mesh refinement**: The AMR++ library for patch based refinement is described in section 7.

Solvers for partial differential equations, such as the OverBlown solver described in section 6, are written using the above classes.

**Array operations**

```c
// Solve u_xx + u_yy = f by a Jacobi Iteration
Range R(0,n) // ... define a range of indices: 0,1,2,...,n
float array u(R,R), f(R,R) // ... declare two two-dimensional arrays
f = 1.; n = 0.; h = 1./n; // ... initialize arrays and parameters
Range I(1,n-1), J(1,n-1); // ... define ranges for the interior

for( int iteration=0; iteration<100; iteration++ )
    u(I,J) = .25*(u(I+1,J)+u(I-1,J)+u(I,J+1)+u(I,J-1)-f(I,J)*(h*h)); // ... data parallel
```

**Adaptive mesh refinement**

Adaptive mesh refinement is the process of permitting local grids to be added to the computational domain and thus adaptively tailoring the resolution of the computational grid. The
block-structured AMR algorithm implemented in Overture provides such support for both simple problems with a single underlying grid, and problems that use the composite overlapping grid method. The AMR algorithm itself uses the multiple grid functionality provided by the basic Overture classes in an essential way. AMR results is greater computational efficiency but is difficult to support. AMR++ is a library within the Overture framework which builds on top of the previously mentioned components and provides support for AMR applications requiring adaptive mesh refinement. AMR++ is current work being developed and supports the adaptive regridding, transfer of data between adaptive refinement levels, parent/child/sibling operations between local refinement levels, and includes parallel AMR support. AMR++ is a parallel adaptive mesh refinement library because it is uses classes which derive their parallel support from the A++/P++ array class library.

Grid Generation

Overture has support for the creation of overlapping grids for complicated geometries. The process of generating an overlapping grid consists of two basic steps. In the first step a number of component grids are generated. Each component grid represents a portion of the geometry. The component grids must overlap but otherwise can be created locally. Overture provides a collection of Mapping classes that can be used to generate component grids including splines, NURBS, bodies of revolution, hyperbolic grid generation, elliptic grid generation, trans-finite interpolation and so on. In addition we are working on methods for reading files generated by CAD programs and generating grids. Figure (2) shows how hyperbolic grid surface grid generation can be used to generate a single smooth grid over a CAD surface described by a collection of trimmed NURBS. This is accomplished with the aid of the SURGRD hyperbolic surface grid generator.$^7$
Given the component grids, the overlapping grid is constructed using the Ogen grid generator. This latter step consists of determining how the different component grids interpolate from each other, and in removing grid points from holes in the overlapping grid domain, and removing unnecessary grid points in regions of excess overlap. Ogen requires a minimal amount of user input. The grids in figure (3) were all created with Ogen.

Writing PDE solvers

The OverBlown framework is being designed to solve the Navier-Stokes equations at all flow speeds, using different algorithms at different Mach numbers. Although the currently distributed version only solves the incompressible Navier-Stokes equations, we are developing a low-Mach number and high Mach number algorithms as well as adding support for chemically reacting flows. Ref 4th order.

Figure (4) shows streamlines of the solution to the incompressible Navier-Stokes equations around a rotating stirring stick and the script file that was used to run OverBlown. Figure (5) shows a result from an incompressible flow computation around a NACA 0012 airfoil. The grid was generated using the elliptic grid generator in Overture. Figure (6) shows results of a three-dimensional computation from OverBlown.

Software availability

The Overture framework and documentation is available for public distribution from the web site, http://www.llnl.gov/casc/overture. The
Figure 3: Sample 2D and 3D overlapping grids generated with the Ogen grid generator.
* Choose the overlapping grid:
stir.hdf
stir.show
incompressibleNavierStokes
turn off twilight zone
project initial conditions
turn on moving grids
specify grids to move
stir
rotate
0. 0. 0.
.5
done
choose grids for implicit
all=explicit
stir=implicit
done
pde parameters
nu
.01
done
boundary conditions
all=noSlipWall
done
initial conditions
uniform flow
p=1.
final time (tf=)
.5	imes

times to plot (tp=)
.025

Figure 4: On the left is a sample script command file for running the flow solver OverBlown. On the right is the result from the computation of incompressible flow around a rotating stirring stick.
OverBlown flow solver is available for limited distribution, please contact Bill Henshaw for further information.

References


[22] Xabier Garaizar, Richard Hornung and Scott


Figure 5: On the left is a sample script command file for running the flow solver OverBlown. On the right is the result from the computation of incompressible flow around a NACA 0012 airfoil.
Figure 6: Incompressible flow through a three-dimensional valve.