OVERTURE: AN OBJECT-ORIENTED FRAMEWORK FOR OVERLAPPING GRID APPLICATIONS

William D. Henshaw
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 on overlapping grids. We describe some of the tools in Overture that can be used to generate grids and solve partial differential equations (PDEs). **Overture** contains a collection of C++ classes that can be used to write PDE solvers either at a high level or at a lower level for efficiency. There are also a number of tools provided with **Overture** that can be used with no programming effort. These tools include capabilities to • repair computer-aided-design (CAD) geometries and build global surface triangulations, • generate surface and volume grids with hyperbolic grid generation, • generate composite overlapping grids, • generate hybrid (unstructured) grids, • solve particular PDEs such as the incompressible and compressible Navier-Stokes equations.

INTRODUCTION

The **Overture** framework is a collection of C++ libraries that provides tools for solving partial differential equations. **Overture** can be used to solve problems in moving geometries using the method of composite 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. At one level **Overture** can be used by the student or researcher who would like write a program to solve a new problem or investigate a new algorithm. The framework provides the basic functionality such as grid generation and difference and interpolation operators that would be very time consuming to develop from scratch. **Overture** can be programmed at a high level as illustrated in the sample code in figure (1). Alternatively, critical parts of a code can be written at a lower level for better performance; it is easy, for example, to call a Fortran subroutine that might implement some numerically intensive algorithm on a single structured grid. **Overture** also provides a collection of tools such as **Ogen** for overlapping grid generation, **Ugen** for building unstructured hybrid grids, **Rap** for repairing CAD geometry and building component grids and **OverBlown** for solving the time dependent incompressible or compressible Navier-Stokes equations with adaptive mesh refinement and moving grids.

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, see for example, Steger et al. [23], Buning et al. [4], Meakin [17, 18], Kiris et.al. [14], Brown et. al.[3], and Henshaw [10]. 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.

In other related work the Chimera Grid Tools from Chan et.al.[5] is a widely used toolkit for the generation of surface and volume grids for overset applications. Other overlapping grid generators include PEGSUS-4 [26], PEGASUS-5 [25], Beggar [15], DCF[16], and xCog/Chalmesh [1920]. Popular PDE solvers for overlapping grids include OVERFLOW [4] and INS3D [22]. There are a number of other very interesting projects developing scientific object-oriented frameworks. These include the SAMRAI framework for structured adaptive mesh refinement [9], PETSc (the Portable Extensible Toolkit for Scientific Computation) [2], POOMA (Parallel Object Oriented Methods and Applications)[13] and Diffpack[1].

One of the goals of **Overture** is to promote the use of overlapping grids as a useful approach for solving a variety of problems. **Overture** has been developed by a small group of applied mathematicians and computer scientists as part of our research into accurate and efficient algorithms for solving PDEs and as such should be considered foremost as a research tool. The source code for **Overture** and **OverBlown** is made freely available at `www.llnl.gov/CASC/Overture`.

SOFTWARE SUMMARY

In this section we summarize the tools and classes provided with **Overture** .

From the point of view of a researcher who would like to develop a computer code to solve a particular problem, **Overture** provides a collection of C++ classes and a set of sample codes that can be used as a starting point for more advanced programs. These classes, summarized here, provide support for geometry and grid generation, arrays, grids and grid functions, operators and graphics.

From the point of view of a user who would like to solve problems without writing computer code, various programs are provided with **Overture** for CAD repair, component grid generation, overlapping grid generation, hybrid grid generation and the solution of the Navier-Stokes equations. These programs are also mentioned below.

The following list is generally ordered so that items earlier in the list do not depend on those later in the list.

Low level utilities

- **GeometricADT**: an alternating digital tree for fast geometric searching.
- **ArraySimple**: light-weight multi-dimensional arrays for C++.

Parallel Arrays

Overture uses the A++/P++ multi-dimensional distributed parallel arrays for C++. These arrays support a fortran90 style array syntax.

Database access

Most objects in **Overture** such as those appearing later in this list know how to *get* and *put* themselves to a data-base file using the low-level functions supplied by the **GenericDataBase**. Thus, for example, when an overlapping grid is saved we do not save the grid points but rather the Mapping that was used to build each component grid. This may save space if the Mapping is defined by an analytic formula; thus the grid points of a 3D box are not saved. When the overlapping grid is read back in the Mapping object is recreated and the grid can be recomputed if it is needed.

- **GenericDataBase**: a generic interface to a hierarchical database.
- **HDF_DataBase**: an implementation of GenericDataBase based on HDF from NCSA.

Utilities

- **TridiagonalSolver**: block tridiagonal solver.
- **OGFunction, OGPolyFunction, OGTrigFunction, OGPulseFunction**: define analytic functions and derivatives used to test PDE solvers with the method of analytic solutions.

Mappings

Most geometry is represented through the Mapping class. Mappings often represent a transformation from the unit *square* in d-dimensions, $[0,1]^d$ to cartesian space in $n-$dimensions, $\mathbb{R}^n$. Mappings are also used to represent other transformations such as translations, rotations and scalings. Mappings include

- **LineMapping, CircleMapping, SquareMapping, AnnulusMapping, BoxMapping, SphereMapping, CylinderMapping, PlaneMapping, QuadraticMapping** : analytic mappings.
- **ComposeMapping**: compose two Mappings in order to rotate, scale shift or stretch grid lines etc.
- **TFIMapping**: define a 2D or 3D grid through transfinite interpolation of boundaries.
- **StretchMapping**: used to stretch grids lines with exponential, inverse hyperbolic tangent, and hyperbolic tangent functions.
- **AirfoilMapping**: a collection of curves including the NACA series of airfoils.
- **SmoothedPolygon**: defined a grid for a 2D polygon with smoothed corners.
- **CrossSectionMapping** : define a surface from cross sections.
- **OffsetShell** : build a grid around a thin shell or plate.
- **NurbsMapping**: define a NURBS (non-uniform rational b-spline) curve or surface with support for editing and repair of trim curves, joining, splitting, interpolating etc.
- **TrimmedMapping**: Trim a mapping that represents a surface, usually used to trim a NurbsMapping.

- **IntersectionMapping**: intersect two Mapping's to determine the curve or points of intersection.

- **JoinMapping**: build a grid to join two intersecting surfaces.

- **FilletMapping**: build a fillet to join two intersecting surfaces.

- **SweepMapping**: sweep or extrude a mapping along a curve.

- **RevolutionMapping**: revolve a mapping about a line.

- **DataPointMapping**: define a mapping defined from collection of data points, a continuous representation is obtained through interpolation.

- **CompositeSurface**: represent a collection of Mappings; usually used to represent a collection of trimmed surfaces from a CAD surface.

- **HyperbolicGridGenerator**: build surface or volume grids by marching.

- **EllipticGridGenerator**: solve the elliptic grid generation equations with multigrid.

- **CompositeTopology**: determine the connectivity of a collection of trimmed patches (CompositeSurface) using an edge matching algorithm and build a global triangulation.

- **UnstructuredMapping**: represent an unstructured grid, optionally build adjacency information. The **project** function can project a point in space onto a surface represented as a unstructured grid.

- **Inverse** : this class provides an inverse (or closest point to a curve or surface) for Mapping's that do not have an inverse defined. It will invert a general logically rectangular mapping using a stencil walk and Newton's method.

- **MappingBuilder**: higher level interface for building surface and volume grids on a CAD surface.

Graphics Interface

- **GL_GraphicsInterface** : defines the interface for plotting results in multiple windows, implemented in OpenGL and Motif.

- **GUIState, DialogData** : build a graphical user interface in C++ with menu's, buttons and dialog windows.

- **Ogshow, ShowFileReader** : save and read solutions, grids and data from a data-base file.

CAD Fixup and Grid Generation

Rap is a high level interface for CAD repair and component grid generation supporting the reading of IGES files, trim curve editing, surface repair, and generating a global triangulation on patched CAD surfaces.

Grids

- **MappedGrid** : represent a grid defined by a Mapping; optionally supplies vertices, cell-centers, Jacobian derivatives, boundary normals, face normals, etc. The grid keeps a pointer to the Mapping that defines it. This means that if the grid needs to be refined the Mapping can be evaluated to precisely determine the location of the new grid points.

- **GridCollection** : a collection of MappedGrid's possibly arranged into a hierarchy for AMR or multigrid.

- **CompositeGrid** : a GridCollection plus the interpolation information required for overlapping grids.

Grid Functions

- **MappedGridFunction**: a grid function (discrete field variable) that can be a vector, tensor etc. The grid function is extended with extra information in order to hold the sparse matrix representation of PDE operator and boundary conditions (for implicit methods for example).

- **GridCollectionFunction**: collections of MappedGridFunction's that can be arranged into a hierarchy for AMR or multigrid.

- **CompositeGridFunction**: a grid function for a **CompositeGrid**.

Operators

Overture provides second and fourth-order finite difference operators, second-order conservative finite-volume operators and a wide variety of boundary conditions including Dirichlet, Neumann, mixed, extrapolation, specify normal or tangential components, extrapolate normal or tangential component, specify normal derivative of the normal or tangential component, etc.

- **MappedGridOperators**: operators for a structured grid.

- **GridCollectionOperators, CompositeGridOperators**: operators for collections of grids.

- **Interpolant**: Arbitrary order overlapping grid interpolation, implicit or explicit.

Higher level graphics operations

- **Grid plotting**: plot grids, AMR levels, multigrid levels.
- **streamLines**: 2D streamlines.
- **contour** : 2D contours, 3D contour cuts, iso-surfaces, coordinate surfaces etc.
- **plotStuff**: graphics post-processor, display solutions on moving and adaptive grids; plot derived functions and derivatives etc.

AMR support

Overture uses the BOXLIB library from LBNL for low-level AMR operations.

- **Regrid**: build the AMR hierarchy of grids from an error measure.
- **ErrorEstimator** : define common error estimators.
- **InterpolatRefinements**: functions for interpolating between coarse and fine levels and for interpolating from an old AMR grid to a new one.
- **Interpolate**: lower level functions for interpolating between patches of differing refinement.

Grid Generation

- **Ogen** overlapping grid generator for cutting holes and finding the interpolation points. See the description later in this paper.
- **Ugen** : hybrid grid generator for generating a grid with structured patches connected by unstructured regions.

Utilities

- **Integrate** : integrate a function on an overlapping grid taking into account the overlap.
- **DataFormats**: read and write files with various other formats such as Plot3d.
- **FileOutput** : output grid and solution information into a ascii file.

Solvers for Boundary Value Problems

- **Ogmg**: multigrid solver for a scalar elliptic equation on overlapping grids; automatic generation of coarse grids and coarse grid equations.
- **Oges**: overlapping grid equation solver for solving boundary value problems; interface to sparse solvers such as Yale, SLAP, PETSc, and **Ogmg** .

OverBlown

A solver for the time dependent Navier-Stokes equations

- **incompressible** Navier-Stokes
- **compressible** Navier-Stokes and Euler equations (Jameson scheme).
- **reacting** Euler equations (simple detonation reactions) (Godunov scheme).
- **RigdBodyMotion** : integrate equations of motion for rigid bodies.

Writing PDE solvers

The C++ program shown in figure (1) is a high-level **Overture** code for solving a convection diffusion equation

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

on an overlapping grid. In this program, the **CompositeGrid** is read in from a data-base file and a grid function u is built and initialized to 1. Note that u represents the solution on a collection of structured grids. Finite difference operators are built and used to advance the solution using a simple forward-Euler time-stepping method. The operator $u.x()$ is the C++ notation for calling the member function "x" of the floatCompositeGridFunction class. It will return a new floatCompositeGridFunction that holds the x-derivative of u on a set of curvilinear grids. At each time step the solution is interpolated (updating the overlapping grid interpolation points) and the boundary conditions are applied. Contours of the solution are interactively plotted.

This high-level code can be made more efficient by replacing the computationally expensive portion (the line `u+=dt*( -a*u.x()-b*u.y()+nu*(u.xx()+u.yy()) );`), either with calls to lower level **Overture** functions or by writing a fortran or C program that advances the solution on a single component grid.

CAD repair and component grid generation

```
int main()
{
    CompositeGrid cg; // create a composite grid
    getFromADataBaseFile(cg,"myGrid.hdf"); // read the grid in
    floatCompositeGridFunction u(cg); // create a grid function
    u=1.; // assign initial conditions
    CompositeGridOperators op(cg); // create operators
    u.setOperators(cg);
    PlotStuff ps; // make an object for plotting
    // — solve a PDE —-
    float t=0, dt=.005, a=1., b=1., nu=.1;
    for( int step=0; step<100; step++ )
    {
        u+=dt*( -a*u.x()-b*u.y()+nu*(u.xx()+u.yy()) );
        t+=dt;
        u.interpolate(); // interpolate overlapping boundaries
        // apply the BC u=0 on all boundaries
        u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
        u.finishBoundaryConditions();
        PlotIt::contour(ps,u); // plot contours of the solution
    }
    return 0;
}
```

Figure 1: Program to solve the PDE $u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$

Figure (2) illustrates the process of building overlapping component grids on a CAD geometry. This capability is a more recent addition to **Overture** . The CAD geometry is usually defined as a patched-surface consisting of a set of sub-surfaces (or patches). A sub-surface may be defined in a variety of ways such as with a spline, B-spline or NURBS. In general the sub-surface will be trimmed, in which case only a portion of the surface will be used, the valid region is defined by trimming curves.

The main steps in the process of building grids on CAD geometries are

- **CAD fixup**: When the CAD geometry is defined from a data file such as IGES, it is usually necessary to repair mistakes in the representation. When the geometry is read in, mistakes are detected in the trimming curves. These can be fixed by editing the curves. It is also possible to edit and change the geometry such as removing or adding patches; see Petersson and Chand [21] for more details.

- **CAD connectivity**: Since an IGES file usually contains no topology information it is necessary to determine how the trimmed surface patches connect to one another. We use an edge-matching algorithm which tries to match the edge curve of a trimmed patch to that of a neighbouring patch; this process will effectively remove most gaps and overlaps between the patches. In some very difficult cases it is necessary to re-edit the geometry. After the patches have been connected a global triangulation is constructed. The global triangulation is used as a first guess when projecting a point onto the original CAD geometry.

- **Surface and Volume Grid Generation**: Structured surface and volume grids can be generated using hyperbolic grid generation. This approach was developed by Steger, Chan and Buning [7, 6] and is also available in Gridgen, see Steinbrenner and Chawner [24]. We have implemented our own version within the Overture framework [12]. The hyperbolic grid generator solves a set of hyperbolic equations to generate a surface grid starting from some initial curve. At each step, the positions of the new grid points are predicted from values of the current grid points and the normal to the surface. Surface grids may be constructed on geometries that are represented either as a structured patch, or an unstructured triangulation or a patched CAD representation. When constructing grids on a CAD geometry the predicted points are projected onto the patched surface by first projecting the point onto the global triangulation

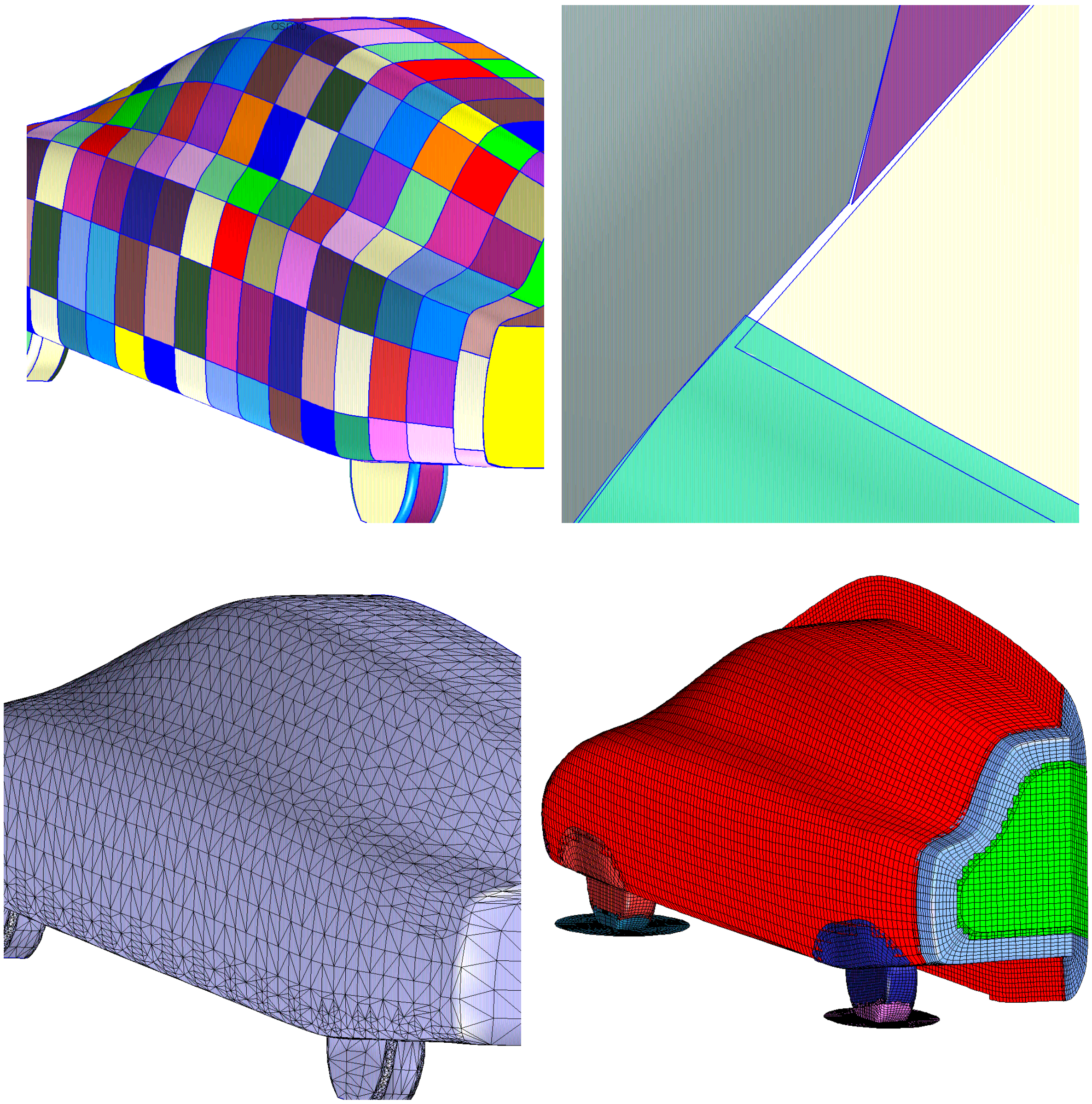


Figure 2: Top left: CAD geometry for a car consisting of a patched surface. Top right: closeup showing a mismatch in the surface patches that needs to be repaired. Bottom left: after the CAD representation is repaired a global triangulation is built. Bottom right: overlapping grid for the geometry. Overlapping surface and volume grids are constructed with a marching algorithm and the grids are connected with the **Ogen** grid generator.

(using a walking algorithm) and then projecting onto the surface patch. A variety of options are available for building starting curves and specifying boundary conditions. For example, the boundary of the surface patch (or an internal grid line) may be constrained to match a specified curve.

Overlapping and hybrid grid generation

The **Ogen** program can be used to construct an overlapping grid from a collection of overlapping structured component grids. It will automatically cut the holes and determine the interpolation points. **Ogen** is descended from the CMPGRD grid generator [8] but uses a substantially different algorithm resulting in a more robust approach. Some of the features include

- a new hole cutting algorithm with a graceful failure mode.
- corrections for boundaries of grids that overlap but do not match; this is an especially troublesome problem when the grids are highly stretched.
- support for higher order discretizations and interpolation.
- support for cell-centred and vertex-centered interpolation.
- optimized algorithms for moving grids.
- support for block structured adaptive mesh refinement.

The basic steps in the overlapping grid algorithm are shown in figure (3).

The unstructured hybrid grid generator **Ugen** begins with a collection of structured grids. Any overlap between the grids is removed, leaving a gap. The gap is filled with an unstructured grid using an advancing front algorithm. In two-dimensions the gap would be filled with triangles, in three-dimensions a single layer of pyramids is generated followed by tetrahedra. The pyramids match the hexahedra to the tetrahedra.

Figures (3,4) shows some overlapping and hybrid meshes generated by the **Ogen** and **Ugen** programs.

OverBlown Navier-Stokes solver

The **OverBlown** flow solver[11] can be used to solve a variety of equations on overlapping grids including:

- the incompressible Navier-Stokes in 2D and 3D; supports moving grids and axisymmetric flow;
- the compressible Navier-Stokes and Euler equations in 2D and 3D (Jameson scheme).
- the reacting Euler equations in 2D; solved with a Godunov scheme and including one-step and chain-branching reaction models for detonations.

Depending on the equations there is support for moving grids and or adaptive mesh refinement. **OverBlown**, however, does not currently have steady state solution algorithms nor does it have turbulence models.

Figure (5) shows some results computed with **OverBlown** .

Acknowledgments: Thanks to the many people who have contributed to **Overture** in particular David Brown, Kyle Chand, Nathan Crane, Petri Fast, Brian Miller, Dan Quinlan, Bobby Phillip, Anders Petersson and Don Schwendeman. This research was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

[1] N. O. AS, *Diffpack home page*, Tech. Rep. http://www.nobjects.com/diffpack, nobjects.com, 1999.

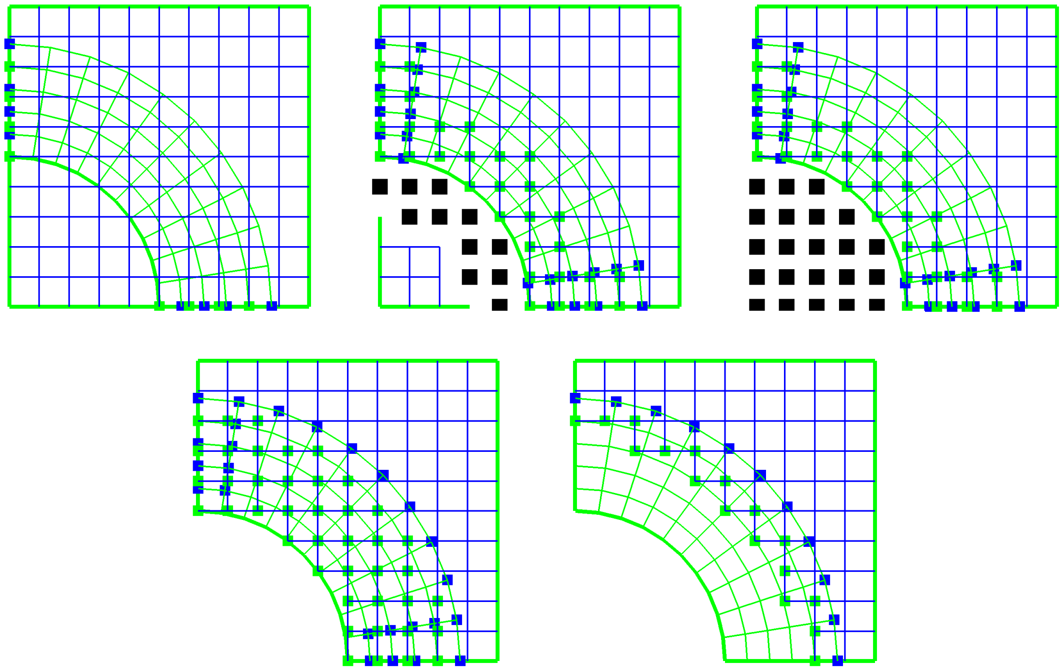
[2] S. BALAY, W. D. GROPP, L. C. MCINNES, AND B. F. SMITH, *The portable extensible toolkit for scientific computation*, Tech. Rep. http://www.mcs.anl.gov/petsc/petsc.html, Argonee National Laboratory, 1999.

[3] D. L. BROWN, *An unsplit Godunov method for systems of conservation laws on curvilinear overlapping grids*, Math. Comput. Modelling, (1994).

[4] P. G. BUNING, I. T. CHIU, S. OBAYASHI, Y. M. RIZK, AND J. L. STEGER, *Numerical simulation of the integrated space shuttle vehicle in ascent*, paper 88-4359-CP, AIAA, 1988.

[5] W. CHAN, S. ROGERS, S. NASH, P. BUNING, AND R. MEAKIN, *User's manual for Chimera Grid Tools*, Tech. Rep. www.nas.nasa.gov/řogers/cgt/doc-/man.html, NASA Ames Research Center, 2001.

[6] W. M. CHAN, *Hyperbolic methods for surface and field grid generation*, in Handbook of Grid Generation, J. F. Thompson, B. K. Soni, and N. P. Weatherill, eds., CRC Press, 1999, ch. 5, pp. 1–26.



Steps in the overlapping grid algorithm, left to right, top to bottom. 1) Boundaries of grids that share an edge with another grid are interpolated, the interpolation points are shown small squares. 2) The grid after cutting holes. Each physical boundary cuts hole points (large squares) in other grids. 3) The hole points are swept out starting from the holes computed in the previous step. 4) All the possible interpolation points are found so that excess overlap can removed or so that better quality interpolation points can be used. 5) The final overlapping grid. The excess overlap has been reduced to the user specified amount.

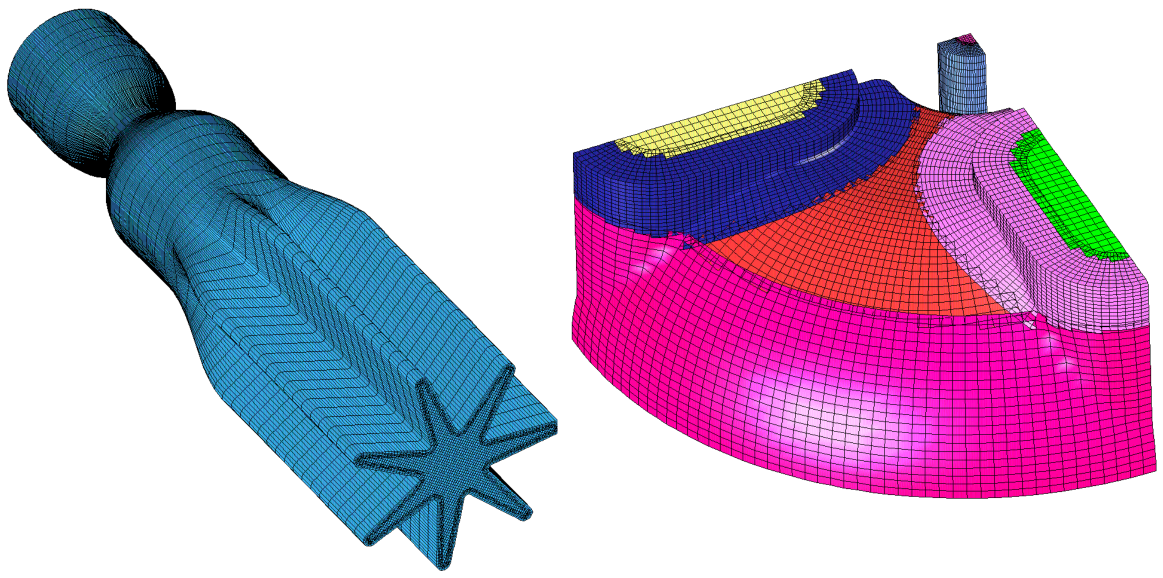


Figure 3: The **Ogen** overlapping grid generator is used to generate overlapping grids given a set of component grids. The steps in the algorithm are shown on top. Two sample grids are shown on the bottom. The geometry for the grid on the left was defined with the capabilities available with **Overture** . The grids on the right were built using hyperbolic surface and volume grid generators on a geometry defined by a CAD package.

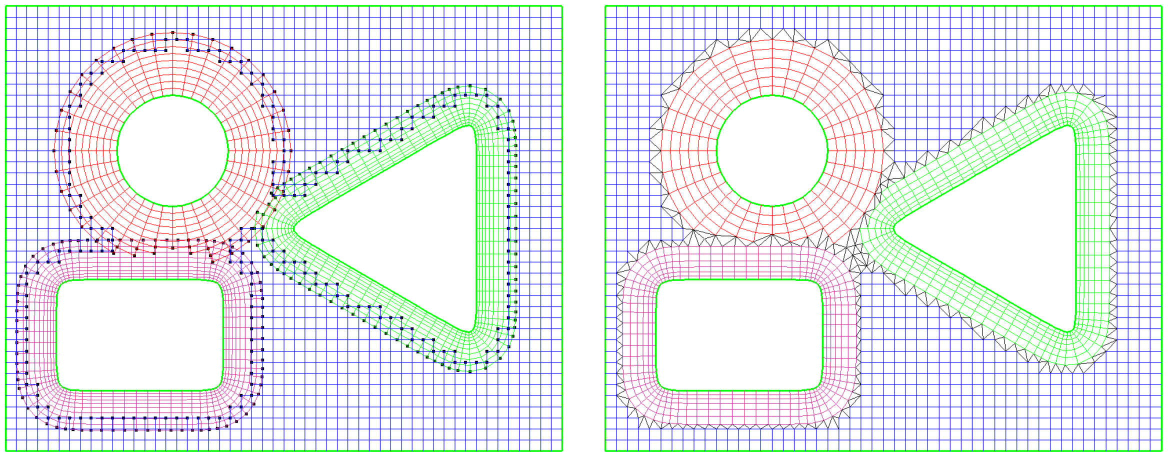


Figure 4: A comparison of an overlapping grid generated by **Ogen** and a hybrid grid generated by **Ugen** .

[7] W. M. Chan and J. L. Steger, *Enhancements of a three-dimensional hyperbolic grid generation scheme*, Applied Mathematics and Computation, 51 (1992), pp. 181–205.

[8] G. Chesshire and W. Henshaw, *Composite overlapping meshes for the solution of partial differential equations*, J. Comp. Phys., 90 (1990), pp. 1–64.

[9] X. Garaizar, R. Hornung, and S. Kohn, *Structured adaptive mesh refinement applications infracture*, Tech. Rep. http://www.llnl.gov/casc/-SAMRAI, Lawrence Livermore National Laboratory, 1999.

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

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

[12] ———, *The Overture hyperbolic grid generator, user guide, version 1.0*, Research Report UCRL-MA-134240, Lawrence Livermore National Laboratory, 1999.

[13] S. Karmesin and et.al, *Parallel object oriented methods and applications*, Tech. Rep. http://www.acl.lanl.gov/PoomaFramework, Los Alamos National Laboratory, 1999.

[14] C. Kiris, D. Kwak, S. Rogers, and I. Chang, *Computational approach for probing the flow through artificial heart devices*, ASME J. of Biofluidmechanical Engineering, 119 (1997), pp. 452–460.

[15] R. Maple and D. Belk, *A new approach to domain decomposition, the beggar code*, in Numerical Grid Genertation in Computational Fluid Dynamics and Related Fields, N. Weatherill, ed., Pineridge Press Limited, 1994, pp. 305–314.

[16] R. Meakin, *A new method for establishing intergrid communication among systems of overset grids*, AIAA paper 91-1586-CP, American Institute of Aeronautics and Astronautics, 1991.

[17] R. L. Meakin, *A DoD CHSSI core project: Scalable implementations of dynamic chimera methods for unsteady aerodynamics*, in Proceedings of the 4th Symposium on Overset Composite Grid and Solution Technology, 1998.

[18] ———, *Composite overset structured grids*, in Handbook of Grid Generation, J. F. Thompson, B. K.

Soni, and N. P. Weatherill, eds., CRC Press, 1999, ch. 11, pp. 1–20.

[19] N. A. PETERSSON, *An algorithm for assembling overlapping grid systems*, SIAM J. Sci. Comp., 20 (1999), pp. 1995–2021.

[20] ——, *Hole-cutting for three-dimensional overlapping grids*, SIAM J. Sci. Comp., 21 (1999), pp. 646–665.

[21] N. A. PETERSSON AND K. K. CHAND, *Detecting translation errors in CAD surfaces and preparing geometries for mesh generation*, in Proceeding of the 10th International Meshing Rountable, 2001.

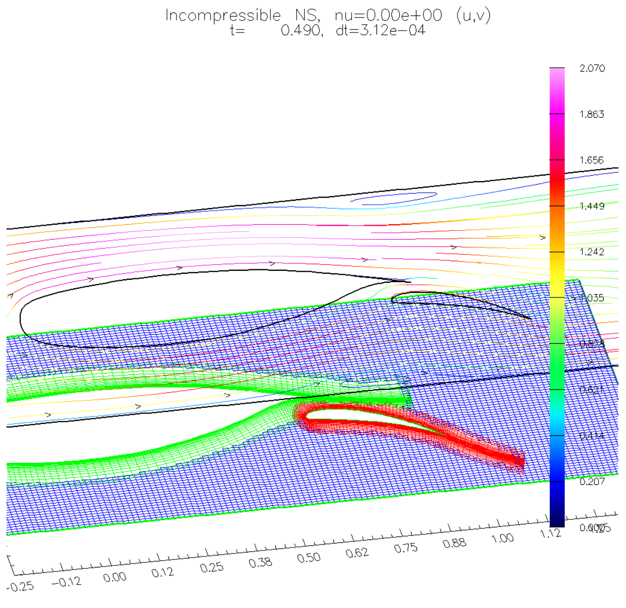
[22] S. ROGERS, D. KWAK, AND C. KIRIS, *Numerical solution of the incompressible Navier-Stokes equations for steady-state and time-dependent problems*, paper 29(4), AIAA, 1991.

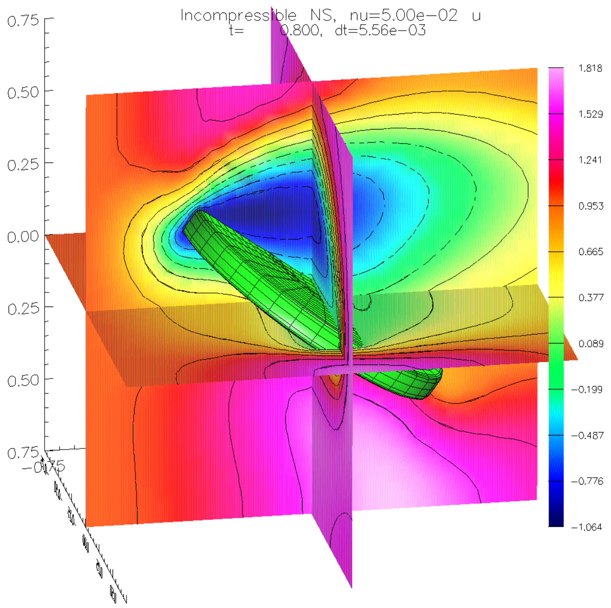
[23] J. L. STEGER AND J. A. BENEK, *On the use of composite grid schemes in computational aerodynamics*, Computer Methods in Applied Mechanics and Engineering, 64 (1987), pp. 301–320.

[24] J. STEINBRENNER AND J. CHAWNER, *Gridgen's implementation of partial differential equation based structured grid generation methods*, in 8th International Meshing Rountable, 1998. www.andrew.cmu.edu/user/sowen/imr8.html.

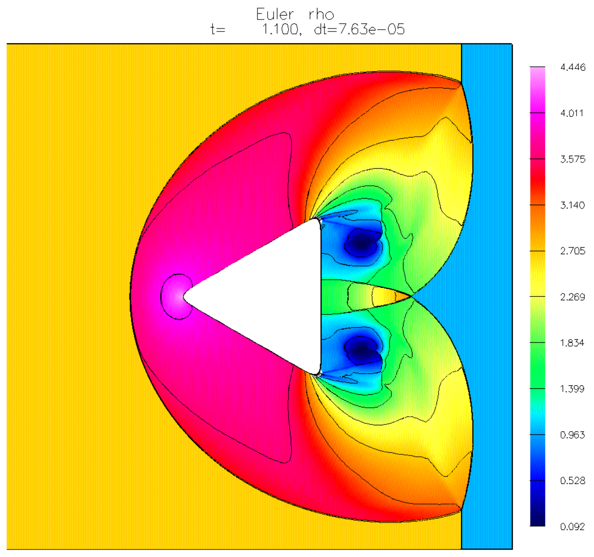
[25] N. SUHS, W. DIETZ, S. ROGERS, S. NASH, AND J. ONUFER, *Pegasus user's guide*, Tech. Rep. www.nas.nasa.gov/r̃ogers/cgt/doc/man.html, NASA Ames Research Center, 2001.

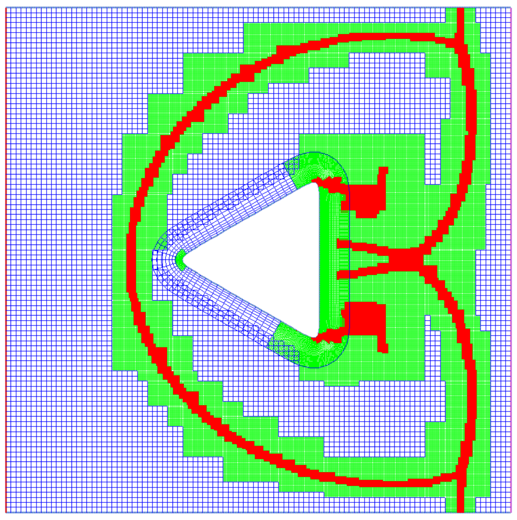
[26] N. SUHS AND R. TRAMEL, *Pegsus 4.0 user's manual*, Research Report AEDC-TR-91-8, Arnold Engineering Development Center, Arnold AFB, TN, 1991.





Solutions of the incompressible Navier-Stokes equations. Left: flow past an airfoil and flap. Right: flow past a rotating disk.





Solution of the Euler equations using adaptive mesh refinement. The solution was computed with two levels of refinement ratio four. Left: contours of the density. Right: the adaptive overlapping grid.

Figure 5: Results from the **OverBlown** solver.