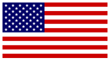
Overture Short Course
Solving PDE's on Overlapping Grids

Bill Henshaw

Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory, Livermore, CA, USA.



Cambridge AMR Workshop,
May, 2011.

Acknowledgments.

Supported by:

ASCR	Department of Energy, Office of Science, ASCR Applied Math Program .
LDRD	LLNL: Laboratory Directed Research and Development (LDRD) program .
NSF	National Science Foundation .

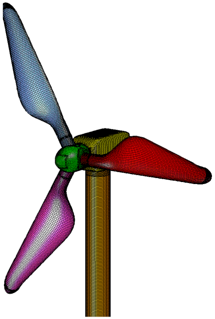
Contributors:

Kyle Chand (LLNL)
Jeff Banks (LLNL)
Prof. Don Schwendeman (Rensselaer Polytechnic Institute)

Increasing computer power is enabling simulations of complex multi-physics applications.

Needed:

1. high fidelity solutions to complex multi-physics problems,
2. in complex moving and deforming geometry,
3. using genuinely high-order accurate and stable methods,
4. for long time integrations.



Example: time dependent LES turbulence simulations to accurately model wind turbines require orders of magnitude more grid resolution than current steady state RANS approximations.

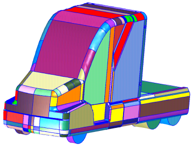
The Overture project is developing PDE solvers for a wide class of continuum mechanics applications.

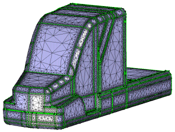
The only open source toolkit for overlapping grids and high-order grid generation.

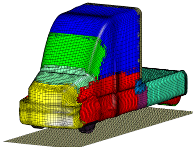
Overture is a toolkit for solving PDE's on overlapping grids and includes CAD, grid generation, numerical approximations, AMR and graphics.

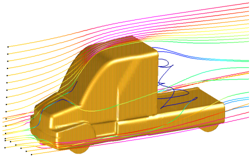
The CG (Composite Grid) suite of PDE solvers (cgcns, cgins, cgmx, cgsm, cgad, cgmp) provide algorithms for modeling gases, fluids, solids and E&M.

Overture and CG represent about 2 million lines of C++ and Fortran; available from www.llnl.gov/CASC/Overture.


Cad fixup


Global triangulation

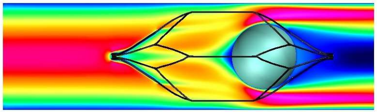
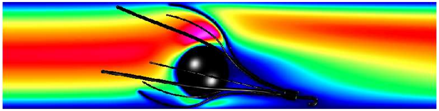

Overlapping grid


Incompressible flow

Overture is used by research groups worldwide

Typical users are graduate students and University/Lab researchers.

- Blood flow and blood clot filters (Dr. Mike Singer, LLNL).
- Flapping airfoils, mircro-air vehicles (Prof. Yongsheng Lian, U. of Louisville).
- Wave-energy devices (Dr. Robert Read, Prof. Harry Bingham, Technical U. of Denmark).
- Plasma physics (Dr. Jeff Banks, Dr. Richard Berger, LLNL).
- Flapping airfoils (Dr. Joel Guerrero, U. of Genoa).
- High-order accurate subsonic/transonic aero-acoustics (Dr. Philippe Lafon, CNRS, EDF).
- Tear films and droplets (Dr. Kara Maki, IMA, and Prof. Richard Braun, U. Delaware).
- Elastic wave equation (Dr. Daniel Appelö, Caltech).
- Compressible flow/ice-formation (Graeme Leese, Prof. Nikos Nikiforakis, U. Cambridge).
- Relativistic hydrodynamics and Einstein field equations (Dr. Philip Blakely, U. Cambridge).
- Converging shock waves, shock focusing (Prof. Veronica Eliasson, USC).
- Pitching airfoils (Dr. D. Chandar, U. of Wyoming, Prof. M. Damodaran, NTU Singapore).
- Hypersonic flows for reentry vehicles, (Dr. Bjorn Sjögreen, LLNL, Dr. Helen Yee NASA).
- Acoustic lens design, shock lithotripsy (Prof. Andrew Szeri, UC Berkeley).
- High-order accurate, compact Hermite-Taylor schemes (Prof. Tom Hagstrom, SMU).
- High-order accurate aero-acoustics (Dr. Ramesh Balakrishnan, ANL).
- Incompressible flow in pumps (Dr. J.P. Potanza, Shell Oil).

Blood flow past wire frame clot filters. M. Singer et.al.

Short Course Summary (1)

An Overview

1. Main features of Overture
2. Overlapping grids.
3. A one-dimensional overlapping grid problem.
4. Approximating derivatives on curvilinear grids.

Using the A++/P++ Array Class

1. Array operations.
2. Parallel array operations.

Short Course Summary (2)

Overview of the Main Overture Classes

1. The Overture graphics interface: windows, menus, dialogs and mouse buttons
2. Mapping's
3. Grid's and GridFunction's
4. Operators
5. Building component grids using the native geometry capabilities

CAD fixup and modification

Fixing and modifying CAD files with mbuilder and rap.

Short Course Summary (3)

Component Grid Generation and CAD

1. Component grid generation on CAD geometries
    1. **mBuilder**: the mapping builder
    2. **hype**: the hyperbolic grid generator

Overlapping Grid Generation

1. **Ogen**: the overlapping grid generator

The Primer examples:

1. MappedGrid examples
2. Overlapping grid examples

Short Course Summary (4)

Adaptive Mesh Refinement and Overlapping Grids

1. Block structured mesh refinement
2. AMR and overlapping grids
3. AMR components of Overture
4. AMR performance and examples

Elliptic PDE Solvers and Implicit Equation Solvers

1. Oges (Overlapping Grid Equation Solver): an interface to
    1. sparse direct solvers (SuperLU).
    2. Krylov based iterative solvers (PETSc).
2. Ogmg: (Overlapping grid multigrid solver):
    1. fast solution of elliptic boundary value problems

Short Course Summary (5)

The CG (Composite-Grid) Suite of PDE solvers:

1. cgad : advection-diffusion solver.
2. cgins : incompressible flow.
3. cgcns : compressible flow with adaptive mesh refinement.
4. cgmx : Maxwell's equations.
5. cgsm : Elastic wave equation (*new*).
6. cgmp : multi-domain multi-physics problems
    1. conjugate heat transfer (fluid flow and solid heat transfer).
    2. fluid-structure interactions (FSI) (*under development*).

Intro Movies

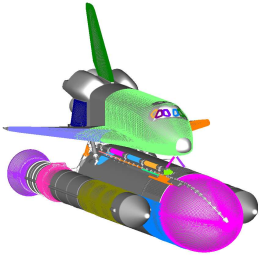
1. shock hitting a collection of (rigid-body) cylinders (Euler with AMR).
2. cylinders falling in a channel (INS with MG).

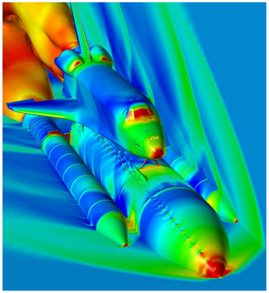
A Short History of *Composite/ Chimera/ Overset/ Overlapping* Grids

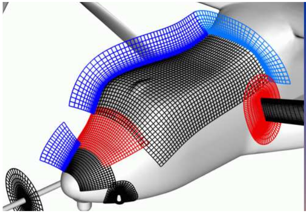
1. Volkov, circa [1966] developed a *Composite Mesh* method for Laplace's equation on regions with piece-wise smooth boundaries separated by corners. Polar grids are fitted around each corner to handle potential singularities.
2. Starius, circa [1977] (student of H.-O. Kreiss) considered *Composite Mesh* methods for elliptic and hyperbolic problems – introduces a hyperbolic grid generator.
3. Steger, circa [1980] independently conceives the idea of the overlapping grid, subsequently named the *Chimera* approach after the mythical Chimera beast having a human face, a lion's mane and legs, a goat's body, and dragon's tail. NASA groups develop grid generator PEGSUS, hyperbolic grid generation and flow solver Overflow (Steger, Benek, Suhs, Buning, Chan, Meakin, et. al.)
4. B. Kreiss [1980] develops overlapping grid generator which subsequently leads to the CMPGRD grid generator [1983] (Chesshire, Henshaw) later leading to the Overture set of tools [1994].

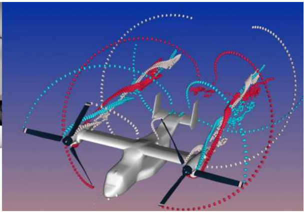
Aerospace applications using overlapping grids.

Nasa's Overflow code has been used for a wide variety of aerospace applications.







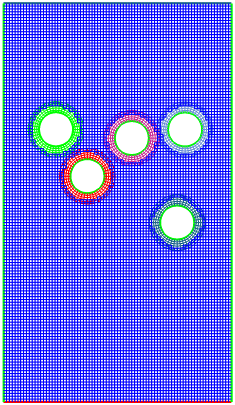


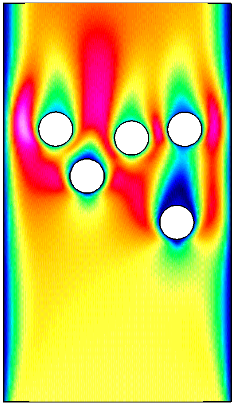
Space shuttle figures courtesy of William Chan and Reynaldo Gomez.
V-22 Osprey figures courtesy of William Chan, Andrew Wissink and Robert Meakin.

What are overlapping grids and why are they useful?

The efficiency of Cartesian grids with the accuracy of boundary fitted grids.

Overlapping grid: a set of structured grids that overlap.





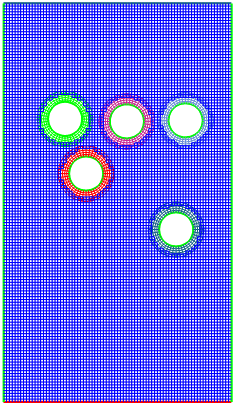
- Overlapping grids can be rapidly generated as bodies move.
- High quality grids under large displacements.
- Cartesian grids for efficiency.
- Smooth grids for accuracy at boundaries (e.g. resolving boundary layers).
- Efficient for high-order methods.

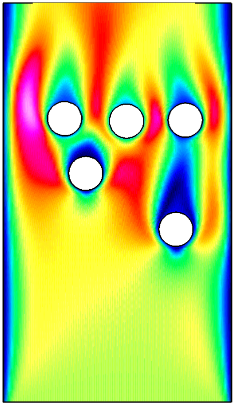
Example: 3D, 4th-order Maxwell : Cartesian grids are 25$\times$ faster than curvilinear.

What are overlapping grids and why are they useful?

The efficiency of Cartesian grids with the accuracy of boundary fitted grids.

Overlapping grid: a set of structured grids that overlap.





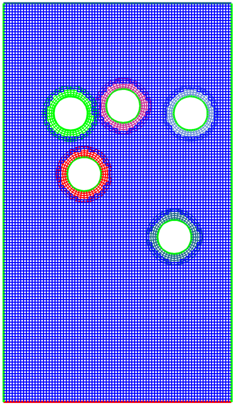
- Overlapping grids can be rapidly generated as bodies move.
- High quality grids under large displacements.
- Cartesian grids for efficiency.
- Smooth grids for accuracy at boundaries (e.g. resolving boundary layers).
- Efficient for high-order methods.

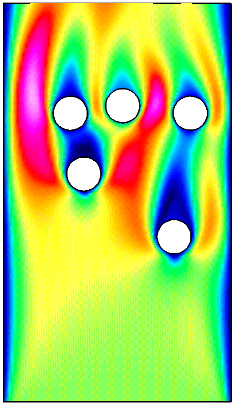
Example: 3D, 4th-order Maxwell : Cartesian grids are 25$\times$ faster than curvilinear.

What are overlapping grids and why are they useful?

The efficiency of Cartesian grids with the accuracy of boundary fitted grids.

Overlapping grid: a set of structured grids that overlap.





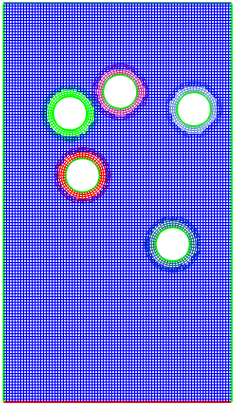
- Overlapping grids can be rapidly generated as bodies move.
- High quality grids under large displacements.
- Cartesian grids for efficiency.
- Smooth grids for accuracy at boundaries (e.g. resolving boundary layers).
- Efficient for high-order methods.

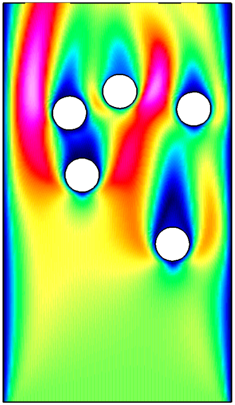
Example: 3D, 4th-order Maxwell : Cartesian grids are 25$\times$ faster than curvilinear.

What are overlapping grids and why are they useful?

The efficiency of Cartesian grids with the accuracy of boundary fitted grids.

Overlapping grid: a set of structured grids that overlap.





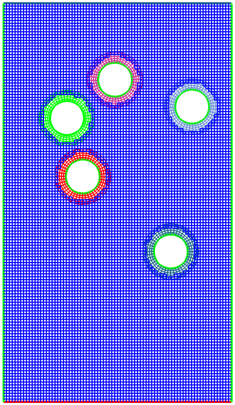
- Overlapping grids can be rapidly generated as bodies move.
- High quality grids under large displacements.
- Cartesian grids for efficiency.
- Smooth grids for accuracy at boundaries (e.g. resolving boundary layers).
- Efficient for high-order methods.

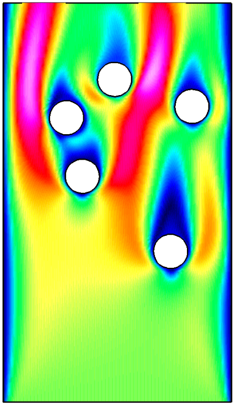
Example: 3D, 4th-order Maxwell : Cartesian grids are 25$\times$ faster than curvilinear.

What are overlapping grids and why are they useful?

The efficiency of Cartesian grids with the accuracy of boundary fitted grids.

Overlapping grid: a set of structured grids that overlap.





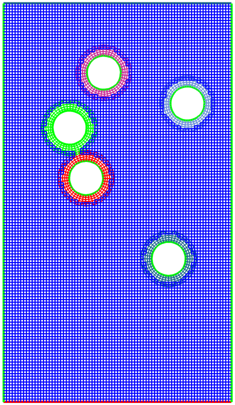
- Overlapping grids can be rapidly generated as bodies move.
- High quality grids under large displacements.
- Cartesian grids for efficiency.
- Smooth grids for accuracy at boundaries (e.g. resolving boundary layers).
- Efficient for high-order methods.

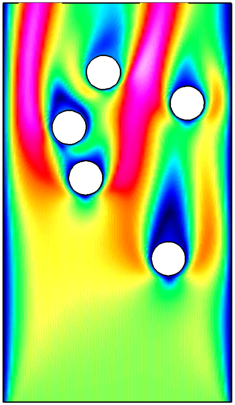
Example: 3D, 4th-order Maxwell : Cartesian grids are 25$\times$ faster than curvilinear.

What are overlapping grids and why are they useful?

The efficiency of Cartesian grids with the accuracy of boundary fitted grids.

Overlapping grid: a set of structured grids that overlap.



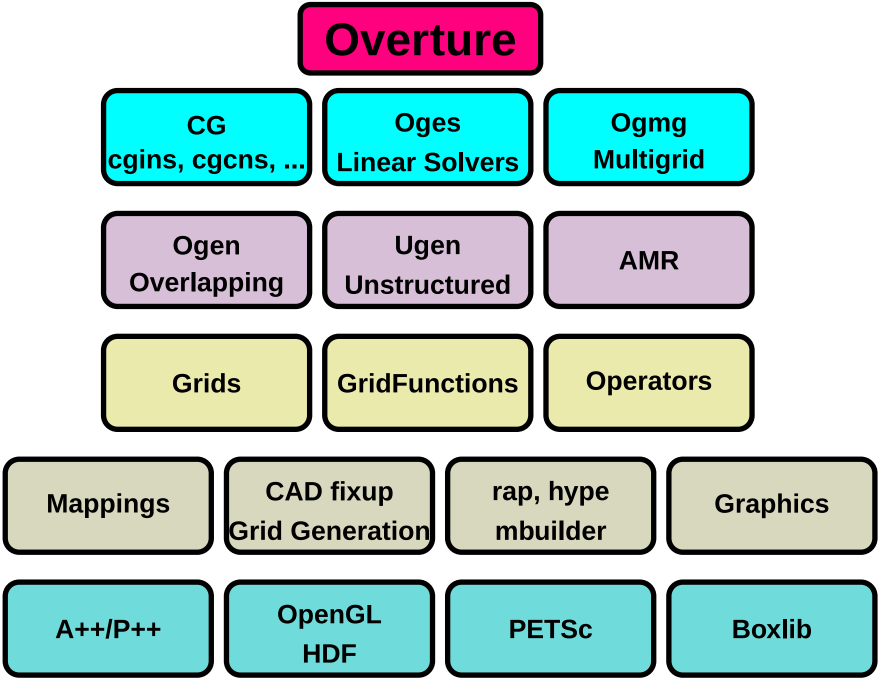


- Overlapping grids can be rapidly generated as bodies move.
- High quality grids under large displacements.
- Cartesian grids for efficiency.
- Smooth grids for accuracy at boundaries (e.g. resolving boundary layers).
- Efficient for high-order methods.

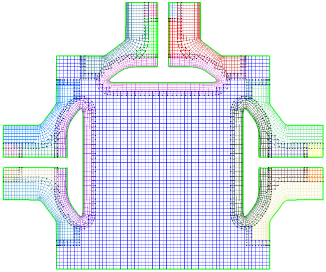
Example: 3D, 4th-order Maxwell : Cartesian grids are 25$\times$ faster than curvilinear.

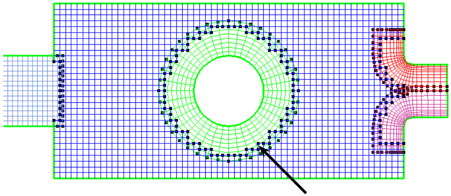
Overture: tools for solving PDE's on overlapping grids

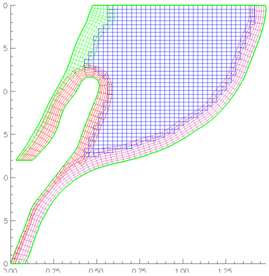
- high level C++ interface for rapid prototyping of PDE solvers.
- built upon optimized C and fortran kernels.
- library of finite-difference operators: conservative and non-conservative, 2nd, 4th, 6th and 8th order accurate approximations.
- support for moving grids.
- support for block structured adaptive mesh refinement (AMR).
- extensive grid generation capabilities (Ogen).
- CAD fixup tools (for CAD from IGES files).
- interactive graphics and data base support (HDF).

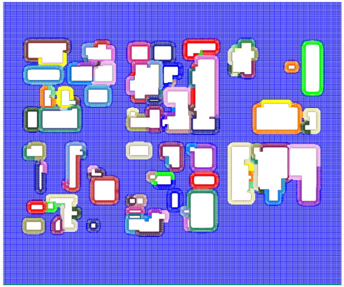

Overture
CG
cgins, cgcns, ...
Oges
Linear Solvers
Ogmg
Multigrid
Ogen
Overlapping
Ugen
Unstructured
AMR
Grids
GridFunctions
Operators
Mappings
CAD fixup
Grid Generation
rap, hype
mbuilder
Graphics
A++/P++
OpenGL
HDF
PETSc
Boxlib

Ogen can be used to build 2D overlapping grids:

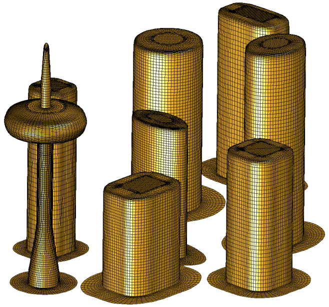
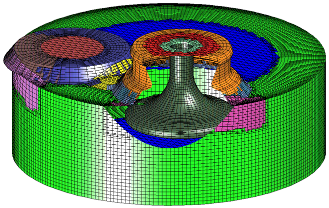
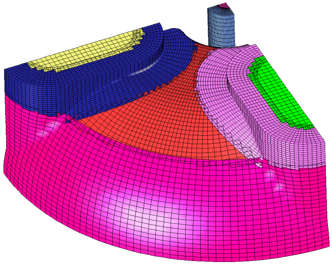
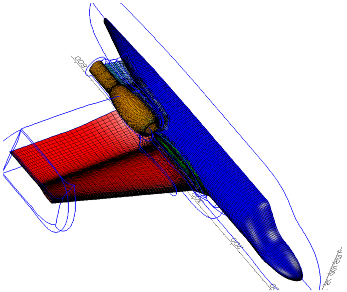






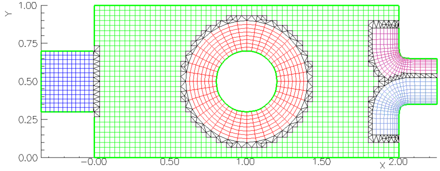


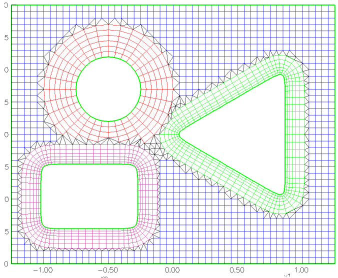
Ogen can be used to build 3D overlapping grids:

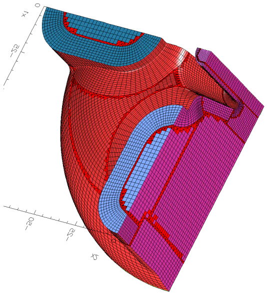





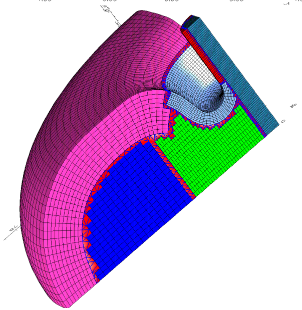
Ugen can generate hybrid (unstructured) grids

Overture has some support for hybrid grids.

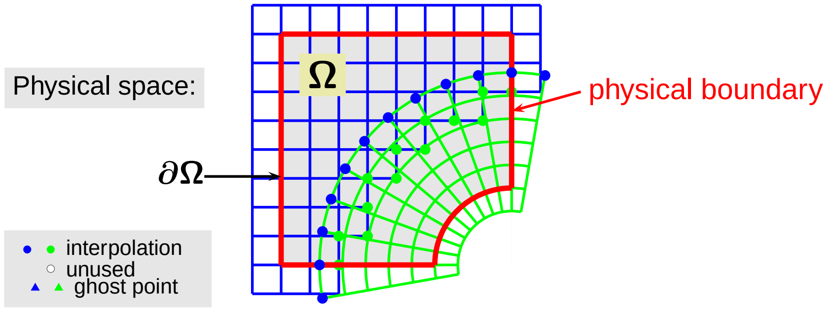








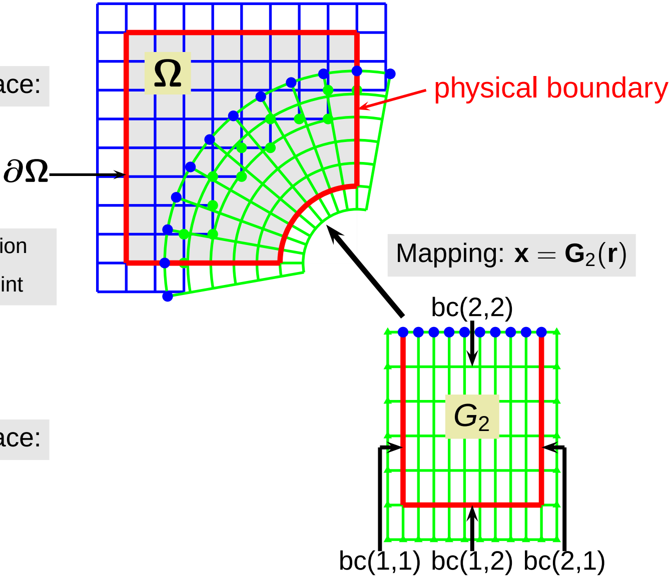
Components of an Overlapping Grid



Components of an Overlapping Grid

Physical space:

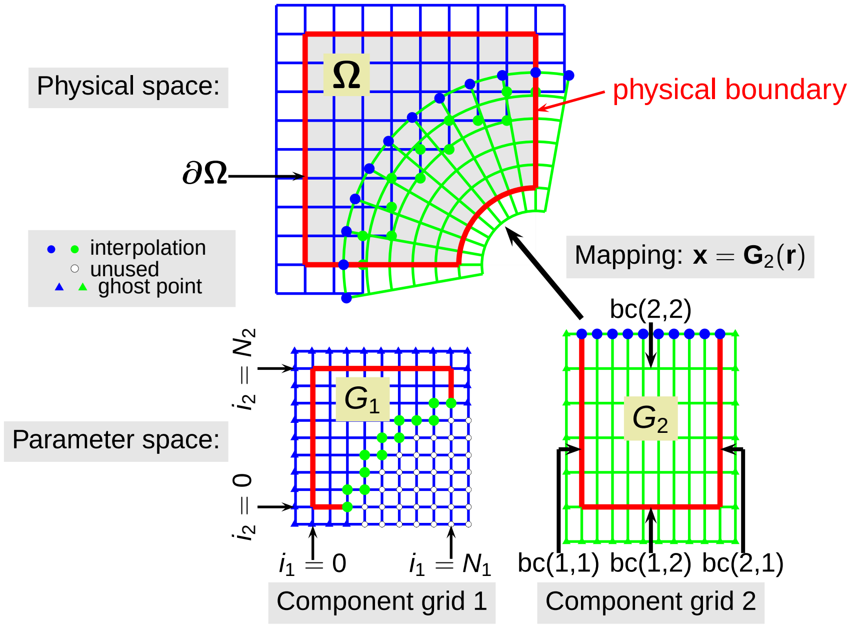
Parameter space:



● ● interpolation
○ unused
▲ ▲ ghost point

Component grid 2

Components of an Overlapping Grid



A one-dimensional overlapping grid example

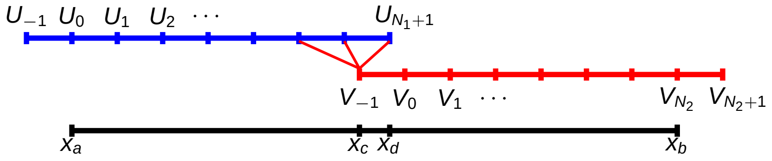
To solve the advection-diffusion equation

$$\begin{aligned} u_t + a u_x &= \nu u_{xx}, && x \in (0,1) \\ u(0,t) = g_0(t), \quad u_x(1,t) &= g_1(t), && \text{(boundary conditions)} \\ u(x,0) &= u_0(x), && \text{(initial conditions)} \end{aligned}$$

introduce grid points on the two overlapping component grids,

$$\begin{aligned} x_i^{(1)} &= x_a + i\Delta x_1, && i = -1,0,1,\ldots,N_1+1, \quad \Delta x_1 = (x_d - x_a)/N_1 \\ x_j^{(2)} &= x_c + (j+1)\Delta x_2, && j = -1,0,1,\ldots,N_2+1, \quad \Delta x_2 = (x_b - x_c)/N_2 \end{aligned}$$

and approximations $U_i^n \approx u(x_i^{(1)}, n\Delta t)$, $V_i^n \approx u(x_i^{(2)}, n\Delta t)$.



Discretize with forward-Euler and central differences

Given the solution at time t^n, compute the solution at time t^{n+1}:

$$(U_i^{n+1} - U_i^n)/\Delta t = -aD_0U_i^n + \nu D_+D_-U_i^n, \qquad i = 1, 2, \ldots, N_1$$

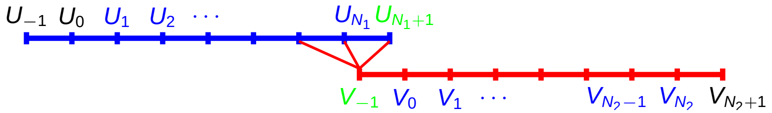
$$(V_j^{n+1} - V_j^n)/\Delta t = -aD_0V_j^n + \nu D_+D_-V_j^n, \qquad j = 0, 2, \ldots, N_2$$

$$U_0^{n+1} = g(t^n), \quad D_0V_{N_2}^{n+1} = g_1(t^{n+1}), \qquad \text{(boundary conditions)}$$

$$U_{N_1+1}^{n+1} = (1-\alpha)(1-\frac{\alpha}{2})\, V_{-1}^{n+1} + \alpha(2-\alpha)\, V_0^{n+1} + \frac{\alpha}{2}(\alpha-1)\, V_1^{n+1}, \quad \text{(interpolation)}$$

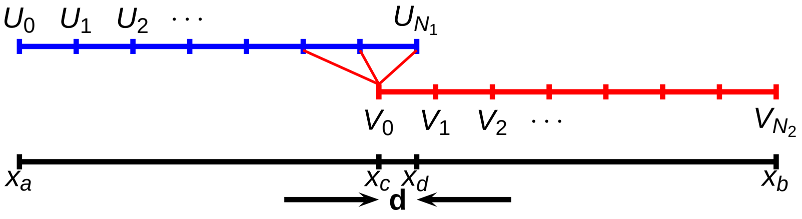
$$V_{-1}^{n+1} = (1-\beta)(1-\frac{\beta}{2})\, U_{N_1-1}^{n+1} + \beta(2-\beta)\, U_{N_1}^{n+1} + \frac{\beta}{2}(\beta-1)\, U_{N_1+1}^{n+1}, \quad \text{(interpolation)}$$

$$D_0U_i^n = \frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x}, \quad D_+U_i^n = \frac{U_{i+1}^n - U_i^n}{\Delta x}, \quad D_-U_i^n = \frac{U_i^n - U_{i-1}^n}{\Delta x}.$$



U_{-1} U_0 U_1 U_2 $\cdots$ U_{N_1} U_{N_1+1}
V_{-1} V_0 V_1 $\cdots$ V_{N_2-1} V_{N_2} V_{N_2+1}

Interpolation between overlapping grids:



U_0 U_1 U_2 $\cdots$ U_{N_1}
V_0 V_1 V_2 $\cdots$ V_{N_2}
x_a x_c x_d x_b
d

When solving Poisson's equation, $\Delta u = f$ with a scheme that is O(h^{2p}) the width of the interpolation formula should be

- width=$2p+1$ if the overlap distance is $d = \mathcal{O}(h)$.
- width=$2p$ if the overlap distance is $d = \mathcal{O}(1)$.

Thus a second-order accurate scheme will normally require 3-point interpolation (quadratic interpolation).

Note: For a first order equation, $u_t + u_x = 0$, 2-point interpolation is sufficient for 2nd-order accuracy when the overlap distance is O(h).

Overture supports a high-level C++ interface

But is built upon mainly Fortran kernels.

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

```
CompositeGrid cg;    // create a composite grid
getFromADataBaseFile(cg,"myGrid.hdf");
floatCompositeGridFunction u(cg);    // create a grid function
u=1.;
CompositeGridOperators op(cg);    // operators
u.setOperators(op);
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()) );    // forward Euler
  t+=dt;
  u.interpolate();
  u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
  u.finishBoundaryConditions();
}
```

Overture supports a high-level C++ interface

But is built upon mainly Fortran kernels.

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

```
CompositeGrid cg;     // create a composite grid
getFromADataBaseFile(cg,"myGrid.hdf");
floatCompositeGridFunction u(cg);     // create a grid function
u=1.;
CompositeGridOperators op(cg);    // operators
u.setOperators(op);
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()) );     // forward Euler
   t+=dt;
   u.interpolate();
   u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
   u.finishBoundaryConditions();
}
```

Overture supports a high-level C++ interface

But is built upon mainly Fortran kernels.

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

```
CompositeGrid cg;     // create a composite grid
getFromADataBaseFile(cg,"myGrid.hdf");
floatCompositeGridFunction u(cg);     // create a grid function
u=1.;
CompositeGridOperators op(cg);    // operators
u.setOperators(op);
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()) );     // forward Euler
   t+=dt;
   u.interpolate();
   u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
   u.finishBoundaryConditions();
}
```

Overture supports a high-level C++ interface

But is built upon mainly Fortran kernels.

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

```
CompositeGrid cg;     // create a composite grid
getFromADataBaseFile(cg,"myGrid.hdf");
floatCompositeGridFunction u(cg);     // create a grid function
u=1.;
CompositeGridOperators op(cg);    // operators
u.setOperators(op);
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()) );     // forward Euler
   t+=dt;
   u.interpolate();
   u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
   u.finishBoundaryConditions();
}
```

Overture supports a high-level C++ interface

But is built upon mainly Fortran kernels.

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

```
CompositeGrid cg;    // create a composite grid
getFromADataBaseFile(cg,"myGrid.hdf");
floatCompositeGridFunction u(cg);    // create a grid function
u=1.;
CompositeGridOperators op(cg);    // operators
u.setOperators(op);
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()) );    // forward Euler
   t+=dt;
   u.interpolate();
   u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
   u.finishBoundaryConditions();
}
```

Spatial approximations to derivatives:

Each grid is defined by a mapping $\mathbf{x} = \mathbf{G}(\mathbf{r})$ from the unit square $\mathbf{r} \in [0,1]^d$ in d-dimensions to physical space $\mathbf{x} \in \mathbb{R}^d$.

Derivatives can be defined by the chain rule (with $\mathbf{r} = (r,s)$, $\mathbf{x} = (x,y)$). For example in two-dimensions:

$$\begin{aligned} u_x &= r_x u_r + s_x u_s \\ \Delta u &= (r_x^2 + r_y^2) u_{rr} + (s_x^2 + s_y^2) u_{ss} + 2(r_x s_+ r_y s_y) u_{rs} + \\ &\quad (r_{xx} + r_{yy}) u_r + (s_{xx} + s_{yy}) u_s \end{aligned}$$

Spatial approximations to derivatives:

Approximations to the derivatives using the *mapping-method* simply approximate the **r** derivatives in the previous. For example, fourth order approximations are

$$u_r \approx D_{0r}(1 - \frac{1}{6}\Delta r^2 D_{+r} D_{-r})U_{i,j}$$
$$u_{rr} \approx D_{0r}(1 - \frac{1}{12}\Delta r^2 D_{+r} D_{-r})U_{i,j}$$

where D_{0r}, D_{+r} and D_{-r} are the central, forward and backward divided difference operators:

$$D_{0r}U_{i,j} = \frac{U_{i+1,j} - U_{i-1,j}}{2\Delta r}, \quad D_{+r}U_{i,j} = \frac{U_{i+1,j} - U_{i,j}}{\Delta r}, \quad D_{-r}U_{i,j} = \frac{U_{i,j} - U_{i-1,j}}{\Delta r}.$$

The *inverse* Jacobian derivatives r_x, r_y, s_x, s_y are given by the mapping. Higher derivatives such as $r_{xx} = (r_x)_x$, $s_{yy} = (s_y)_y$ are approximated in the same manner as for u.

Spatial approximations to derivatives:

Conservative or finite-volume type discretizations are based on the self-adjoint form

$$\nabla\cdot\mathbf{u} = \frac{1}{J}\Big(\frac{\partial}{\partial r}(J\nabla_{\mathbf{x}} r\cdot\mathbf{u}) + \frac{\partial}{\partial s}(J\nabla_{\mathbf{x}} s\cdot\mathbf{u})\Big),$$
$$J = \det(\partial\mathbf{x}/\partial\mathbf{r}) \qquad \text{(Jacobian)}$$

where $\mathbf{u} = (u,v)$ and

$$\nabla_{\mathbf{x}} r\cdot\mathbf{u} = \frac{\partial r}{\partial x}u + \frac{\partial r}{\partial y}v,$$
$$\nabla_{\mathbf{x}} s\cdot\mathbf{u} = \frac{\partial s}{\partial x}u + \frac{\partial s}{\partial y}v.$$

Some References: Theory of Numerical Methods

The stability and accuracy theory for finite difference approximations to initial-boundary-value problems for overlapping grids is primarily founded upon the well established theory for finite difference methods for a single grid. Stability theory is often divided into methods based on *energy estimates* and methods based on *mode analysis (GKS theory)*. For a discussion of energy-estimates and GKS theory see, for example,

- Gustafsson, Kreiss and Oliger, *Time Dependent Methods and Difference Methods*, Wiley, 1995.
- Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth and Brooks/Cole, 1989.

In many cases, the analysis of a nonlinear, variable-coefficient PDE in general geometry can be reduced to the consideration of a constant-coefficient half-plane problem. For a discussion of well-posed problems and this reduction see, for example,

- Kreiss and Lorenz, *Initial-Boundary Value Problems and the Navier-Stokes Equations*, "Academic Press, 1989.

References: overlapping grids

For a general discussion of the overlapping grid approach as well as issues related to the order of accuracy of interpolation see

- G. Chesshire and W.D. Henshaw, *Composite Overlapping Meshes for the Solution of Partial Differential Equations* JCP, 1990.

Early references to overlapping grids

- G. Starius, *Composite Mesh Difference Methods for Elliptic and Boundary Value Problems*, Numer. Math., 1977.
- G. Starius, *On Composite Mesh Difference Methods for Hyperbolic Differential Equations*, Numer. Math., 1980.
- J. L. Steger and J. A. Benek, *On the use of Composite Grid Schemes in Computational Aerodynamics*, Computer Methods in Applied Mechanics and Engineering, 1987.

A++/P++ Arrays...

A++/P++ : Multidimensional Arrays for C++

1. Fortran90/matlab style array operations.
2. Serial and distributed parallel arrays.
3. Code is compiled with A++ header files to run on serial machines.
4. Code is compiled with P++ header files to run on parallel machines.
5. Arrays can be passed to Fortran subroutines (Fortran storage order)

Documentation: See the Overture web page for the manual and quick reference guide.

A++/P++ Clases

1. class Index: Index I(base,count,stride) : for indexing arrays
2. class Range: Range R(base,bound) : for dimensioning (or indexing arrays)
3. class intArray, floatArray, doubleArray : multidimensional arrays (maximum 6 dimensions) distributed across selected processors (also referred to as intDistributedArray, floatDistributedArray, doubleDistributedArray).
4. intSerialArray, floatSerialArray, doubleSerialArray: serial arrays, duplicated across all processors
5. class Partitioning_Type partition : for defining a parallel distribution (set of processors, number of parallel ghost).

A++ : Stencil operations and scalar indexing.

```
#include "A++.h"
...
Range R(0,9);                              // define a Range R={0..9}
doubleSerialArray a(R,R), b(10,10);  // declare and dimension arrays
Index I(1,8), J(1,8);                      // define two Index's: I={1,2,...,8}
b=1.;                                      // array assignment

// Stencil operation:
// (Note: b(I+1,J) is a new array that is a "view" of b).
a(I,J) = .25*( b(I+1,J) + b(I,J+1) + b(I-1,J) + b(I,J-1) );

// Stencil operation by scalar indexing:
for( int j=J.getBase(); j<=J.getBound(); j++ )
for( int i=I.getBase(); i<=I.getBound(); i++ )
{
  a(i,j) = .25*( b(i+1,j) + b(i,j+1) + b(i-1,j) + b(i,j-1) );
}
```

A++: Using the where statement.

```
doubleSerialArray x(10,10), u(10,10), v(10,10);
Range I(1,8), j(1,8);
...
where( x(I,J)>0.5 )
{
  u(I,J)=sin(2.*Pi*x(I,J));
  v(I,J)=cos(2.*Pi*x(I,J));
}

// Equivalent operation with scalar indexing:
for( int j=J.getBase(); j<=J.getBound(); j++ )
for( int i=I.getBase(); i<=I.getBound(); i++ )
{
  if( x(i,j)>0.5 )
  {
    u(i,j)=sin(2.*Pi*x(i,j));
  }
  if( x(i,j)>0.5 )
  {
    v(i,j)=cos(2.*Pi*x(i,j));
  }
}
```

A++: Indirect addressing

Accessing a random set of array elements.

```
doubleSerialArray u(10), v(3);
intSerialArray ia(3);          // holds indirect array indicies

ia(0)=2; ia(1)=4; ia(2)=7;

v(I) = u(ia) + 2.*u(ia+1);


// Equivalent operation with scalar indexing:
for( int i=I.getBase(); i<=I.getBound(); i++ )
{
  v(i) = u(ia(i)) + 2.*u(ia(i)+1);
}
```

A++: Passing arrays to Fortran

A++ arrays are stored in Fortran order.

C++ code:

```
extern "C" { void myfortran_( int& m, int& n, real &u ); }
...
floatSerialArray u(20,10);
// call a Fortran routine:
myfortran_(u.getLength(0),u.getLength(1),*u.getDataPointer());
```

Fortran routine:

```
subroutine myFortran( m,n,u )
integer m,n,i,j
real u(m,n);

do i=1,m
do j=1,n
  u(i,j) = i + 3.*j
end do
end do

return
end
```

A++ : Fast C-style Scalar Indexing

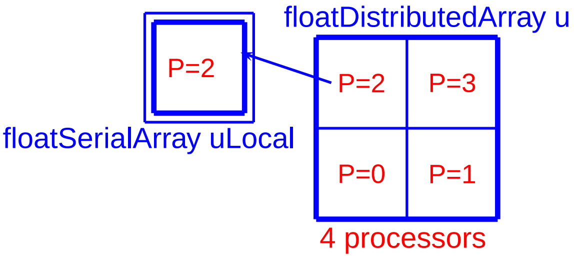
Can be much faster than C++ scalar indexing.

We define a C macro to access the A++ array.

```
#include "A++.h"
...
doubleSerialArray u(10,10);     // declare and dimension arrays
Index I(1,8), J(1,8);           // define two Index's: I={1,2,...,8}

real *up = u.Array_Descriptor.Array_View_Pointer2;
const int uDim0=u.getRawDataSize(0);
const int uDim1=u.getRawDataSize(1);
#define U(i0,i1,i2) up[i0+uDim0*(i1+uDim1*(i2))]

// Stencil operation by C-style scalar indexing:
for( int j=J.getBase(); j<=J.getBound(); j++ )
for( int i=I.getBase(); i<=I.getBound(); i++ )
{
  U(i,j) = .25*( U(i+1,j) + U(i,j+1) + U(i-1,j) + U(i,j-1) );
}
```

P++ : parallel multi-dimensional arrays

1. Arrays are distributed across processors using a partitioning object.
2. A serial array (local array) can be accessed from each processor.
3. Internal ghost-boundaries are added on local arrays.
4. P++ uses Multiblock PARTI (A. Sussman, G. Agrawal, J. Saltz) for ghost boundary updates.
5. Caveat: Best results are obtained by operating on the local arrays and explicitly calling *updateGhostBoundaries* to update the ghost boundaries.



P++ : Sample Code

```
#include "A++.h"
...
Range P(1,5); // Range of processors to use
Partitioning_Type partition(P); // define parallel distribution
partition.SpecifyInternalGhostBoundaryWidths(1,1);

// build a distributed array:
doubleDistributedArray u(100,100,partition);

// access the local array:
doubleSerialArray & uLocal = u.getLocalArrayWithGhostBoundaries();

// operate on local array
uLocal = cos(uLocal)+5.;

// call fortran with local array:
myFortranRoutine(*uLocal.getDataPointer(),...);

// update ghost boundaries between processors
u.updateGhostBoundaries();
```

Overture Classes

1. Graphics
2. Mappings
3. CAD and CAD repair/modification
4. Grids
5. GridFunctions
6. Operators
7. Manufactured solutions (aka *twilight-zone* functions)

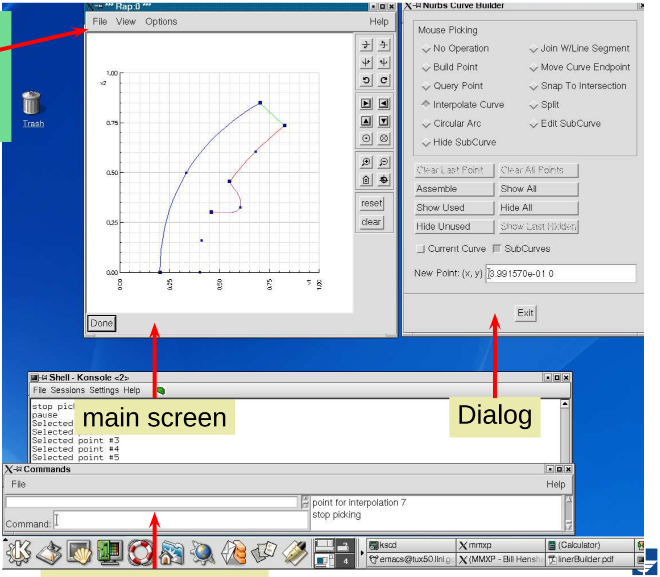
The Overture Graphics Interface

- Built upon OpenGL and Motif
- Mouse driven rotation, zoom, and picking (selection).
- High-level routines for plotting grids, contours, stream-lines.
- Interactive plotting by C++ calls.
- User defined menus and dialogs.
- Program control remains generally with the application, the GUI *event loop* is entered only when input in desired.
- OpenGL calls are restricted to the Graphics Interface classes. Motif is not exposed, usage is restricted to a few functions.

Overture Graphics Interface

File: hardcopy
View: clipping
Options: axes, colour-bar

Mouse Buttons
left : pick
middle : zoom
right : pop-up menu
shift-left : translate
shift-middle : rotate



[✓ Interpolate Curve]

New Point: (x, y) 3.991570e-01 0

```
stop pick
pause
Selected
Selected
Selected point #3
Selected point #4
Selected point #5
```

```
point for interpolation 7
stop picking
```

Command:

main screen

Dialog

command window

Features of the graphical user interface

1. User defined dialog windows that can contain pulldown menus with push or toggle buttons, option menus, text labels (for inputting strings), push buttons, and toggle buttons.
2. Multiple graphics windows and a single command window with a scrollable sub-window for outputting text, a command line, and a scrollable list of previous commands.
3. Rotation buttons , , ,... which rotate the object on the screen about fixed x, y, and z axes (the x axis is to the right, the y-axis is up and the z-axis is out of the screen).
4. Translation buttons , , , ,... which shift the object on the screen along a given axis.
5. Push buttons for making the objects bigger: , or smaller: , and a reset button: to reset the view point, and a **clear** button to erase all objects on the screen.

Features of the graphical user interface (cont'd)

1. A push button ⌖ to set the rotation center.
2. A **rubber band zoom** feature.
3. Mouse driven **translate, rotate and zoom**.
4. A pop-up menu that is active on the command and graphics windows. This menu is defined by the application (= user program).
5. Static pull-down menus (**file**, **view**, and **help**) on the graphics windows. Here, the screen can be saved in different formats, clipping planes and viewing characteristics can be set, annotations can be made (not fully implemented), and some help can be found.
6. Static pull-down menus (**file** and **help**) on the command window. Here command files can be read/saved, new graphics windows can be opened, the window focus can be set, and the application can be aborted.
7. A file-selection dialog box
8. The option of typing any command on the command line or reading any command from a command file. All commands can be entered in this fashion, including any pop-up or pull-down menu item or any of the buttons, **x+r**, **y-r**, **x+**, **y+**, **bigger**, etc.
9. Recording or retrieving a command sequence in a command file.

Geometry, Grids and GridFunctions: Overview

1. Mapping : used to define continuous transformations. Example: the transformation from the unit square to an annulus.
2. MappedGrid: defines a grid for a Mapping; contains the grid points, Jacobian derivatives etc.
3. {int,float,double}MappedGridFunction : holds field values on a MappedGrid. Example: the density or temperature at each point on the grid. Derived from an A++/P++ array.
4. GridCollection : a list of MappedGrid's. Example: the grids forming an adaptive mesh refinement level.
5. {int,float,double}GridCollectionFunction : a list of MappedGridFunction's.
6. CompositeGrid : a GridCollection with interpolation information that represents an overlapping grid.
7. {int,float,double}CompositeGridFunction: holds field values on a CompositeGrid.

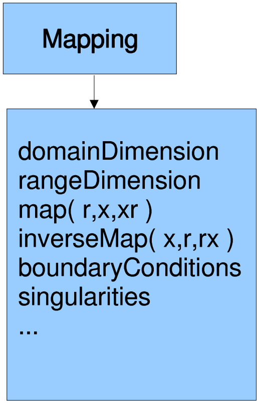
Overview of the Geometry Capabilities in Overture

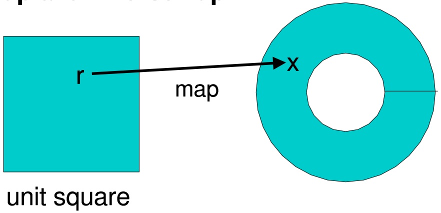
Native Geometry capabilities

1. Curves, surfaces and volumes are represented with the **Mapping** class
2. Many analytic representations including rectangle, annulus, cylinder and sphere
3. Spline and NURBS representations are available.
4. Transfinite interpolation can be used to interpolate curves or surfaces to form 2D-regions or 3D volumes.
5. Transformations such as rotation, scaling, translation, body-of-revolution, stretching-grid-lines, sweeping, and extruding are also represented as a **Mapping**.
6. Mappings can be composed: e.g. compose a annulus-mapping with a stretch-mapping to get a new mapping for an annulus with clustered grid-lines.
7. 3D surfaces can be intersected to form one or more curves of intersection.

A Mapping defines a continuous transformation

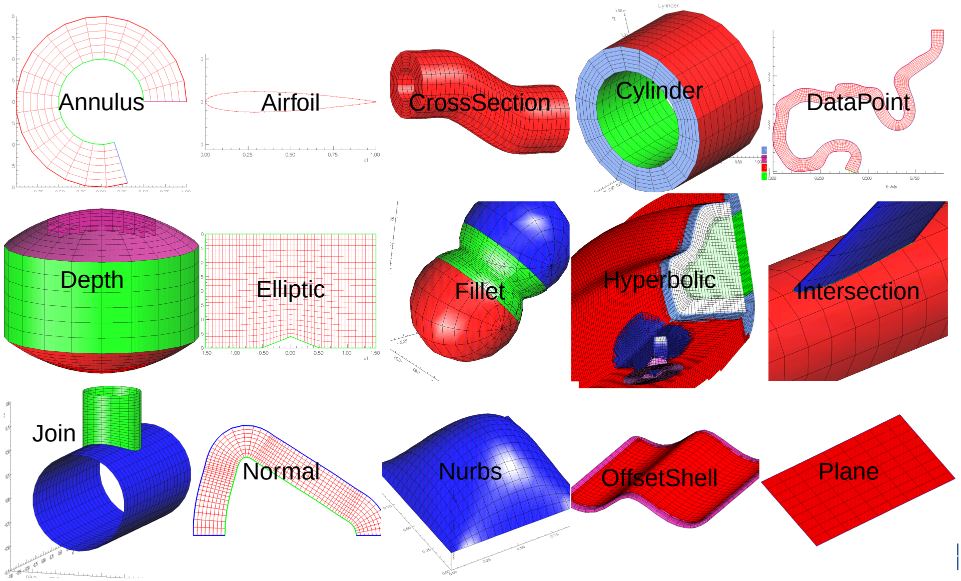
Each mapping has an optimized **map** and **inverseMap**



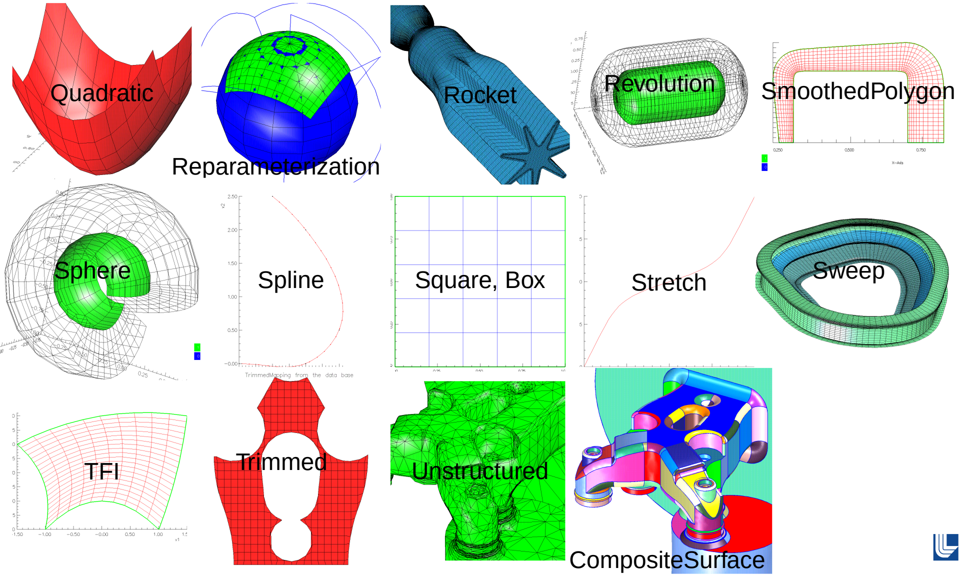


SquareMapping
AnnulusMapping
SphereMapping
HyperbolicMapping
EllipticTransform
MatrixMapping
RevolutionMapping
etc.

Overture Mappings (Part I)



Overture Mappings (Part II)



Demo: Creating Mappings...

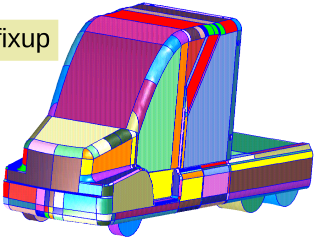
1. show different types of mappings
2. composition, stretching, body of revolution...

Repairing and Modifying CAD Geometry

1. Overture can read the IGES CAD file format.
2. Geometry is usually defined as a B-Rep (boundary representation) consisting of a collection of patches. Each patch is often a trimmed NURBS (non-uniform rational b-spline).
3. There are problems that may exist in the CAD representation such as gaps or overlaps between patches and mistakes in trimming curves.
4. The Overture rap program (an outcome from the Rapsodi project) or mbuilder can be used to fix and modify the CAD geometry.
5. A *global triangulation* is constructed that defines a water-tight surface.
6. The CAD B-Rep and global-triangulation are used when building structured surface grids.

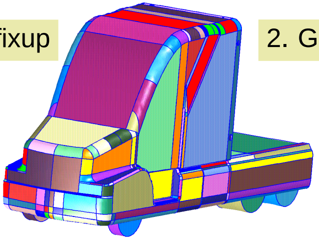
From CAD to Mesh to Solution with Overture

1. Cad fixup

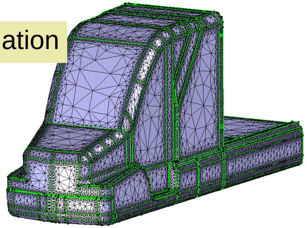


From CAD to Mesh to Solution with Overture

1. Cad fixup

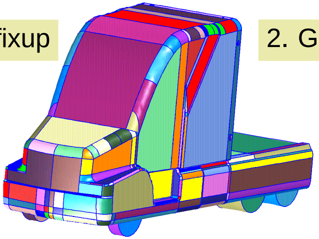


2. Global triangulation

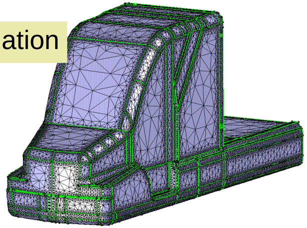


From CAD to Mesh to Solution with Overture

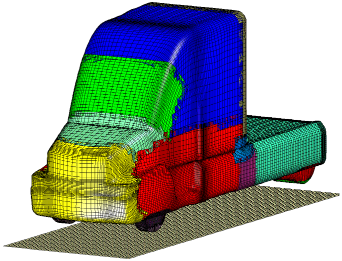
1. Cad fixup



2. Global triangulation

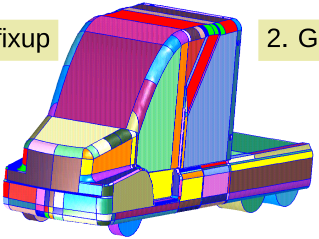


3. Overlapping grid

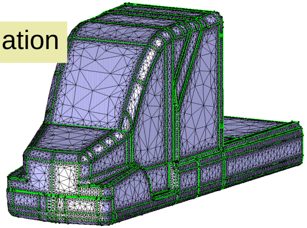


From CAD to Mesh to Solution with Overture

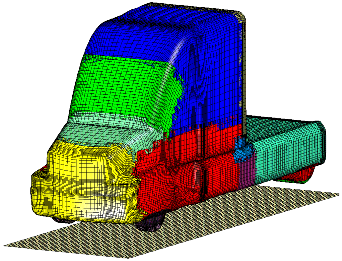
1. Cad fixup



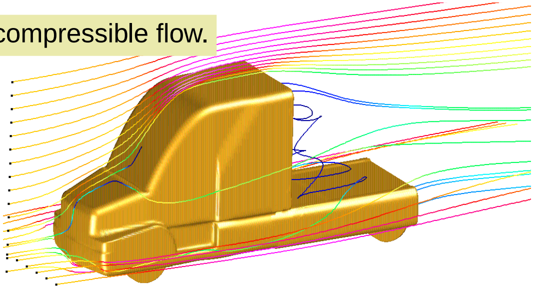
2. Global triangulation



3. Overlapping grid



4. Incompressible flow.



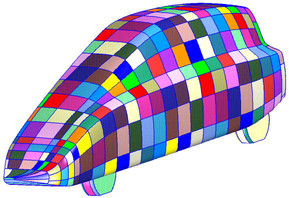
Demo: CAD clean-up for the ASMO car.

The first step in building a grid for the ASMO car is to fix the CAD geometry.

rap asmoNoWheels.cmd

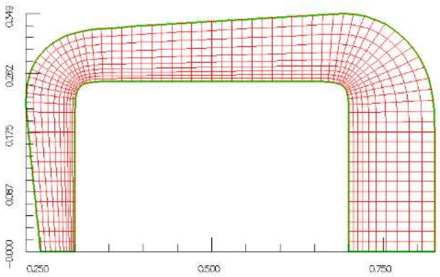
1. clean up a CAD geometry for the *asmo* model car.
2. generate the topology (fix gaps and overlaps between patches).
3. build a water-tight surface triangulation.

(See Overture/sampleGrids/README.)



Grids and Grid Functions

A MappedGrid holds the grid points and other geometry info

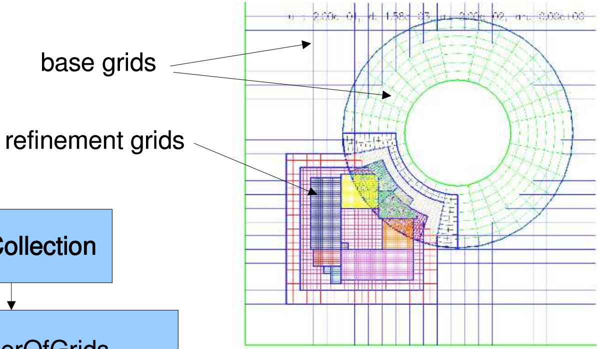


MappedGrid

gridIndexRange
Mapping ← Mapping that defines the geometry
vertex ← grid points
vertexDerivative ← jacobian derivatives
cellVolume
faceNormal
...

optionally computed geometry arrays

A GridCollection holds a list of MappedGrids



GridCollection

numberOfGrids
operator [int g] ← access to a MappedGrid, g=0,1,..

refinementLevel[int l] ← GridCollection for a refinement level
...

A MappedGridFunction holds solution values

realMappedGridFunction

derived from an A++ realArray
which implies it inherits all A++ operators

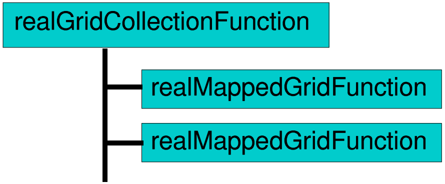
numberOfComponents ← scalar, vector, 2-tensor, ..
MappedGrid
MappedGridOperators

x,y,z,xx,... derivatives

Grid functions can be vertex-centred, cell-centred, face-centred etc.

```
MappedGrid mg(mapping);
realMappedGridFunction u(mg);
u=1.;
```

A GridCollectionFunction is a list of MappedGridFunctions



```
GridCollection gc(...);
Range all;

realGridCollectionFunction u(gc,all,all,all,2);

u=1.;

for( int grid=0; grid<gc.numberOfGrids(); grid++ )
  u[grid]=1.;
```

Overview of the Operator Classes in Overture

Operators define discrete approximations to derivatives and boundary conditions.

1. approximations for ∂_x, ∂_y, ∂_z, ∂_{xx}, ,.., Δ, $\nabla \cdot (a\nabla)$.
2. orders of accuracy 2,4,6,8.
3. finite difference and finite volume approximations.
4. operators: explicitly evaluate derivatives.
5. operators: form the sparse matrix.
6. Boundary conditions: Dirichlet, Neumann, Mixed, ...

Operators can be used at different levels

1. CompositeGridFunctions: $u.x()$ – all points on all grids.
2. MappedGridFunctions: $u.y()$ – all points on one grid.
3. function call: `derivative(u,xOperator,...)` (most efficient).
4. macro for evaluation in C or Fortran codes.

Operators: code examples.

```
SphereMapping sphere;              // Mapping for a sphere
MappedGrid mg(sphere);             // Grid for a sphere

MappedGridOperators op(mg);        // build operators

realMappedGridFunction u(mg), v(mg), coeff(9,all,all,all);

// compute derivatives explicitly:
v= u.x() + 5.*u.y();

// form the sparse matrix
coeff = op.laplacianCoefficients();
```

Operators: efficient evaluation of operators.

```
CylinderMapping cyl;          // Mapping for a cylinder
MappedGrid mg(cyl);           // Grid for a cylinder

MappedGridOperators op(mg);     // build operators

// Grid function with 3 components
realMappedGridFunction u(mg,all,all,all,3);
realSerialArray uLocal = u.getLocalArrayWithGhostBoundaries();

// eval derivatives at interior and boundary points:
Index I1,I2,I3;
getIndex(mg.gridIndexRange(),I1,I2,I3);

// Restrict Index bounds to this processor:
bool ok=ParallelUtility::getLocalArrayBounds(u,uLocal,I1,I2,I3);
if( ok )
{
  // evaluate the x and y derivatives on this grid:
  realSerialArray ux(I1,I2,I3,3), uy(I1,I2,I3,3);
  Range N=3;
  op.derivative(MappedGridOperators::xDerivative,uLocal,ux,I1,I2,I3,N);
  op.derivative(MappedGridOperators::yDerivative,uLocal,uy,I1,I2,I3,N);

}
```

Testing with manufactured solutions.

Known as *Twilight-Zone* forcing in Overture. A very useful technique!

Given a PDE boundary value problem

$$L(u_t, u_x, u_y, \ldots) = F(\mathbf{x}, t)$$

one can create an *exact* solution, $U(\mathbf{x}, t)$ by choosing

$$F(\mathbf{x}, t) = L(U_t, U_x, U_y, \ldots)$$

The Overture OGFunction class defines a variety of exact solutions and their derivatives to support the method of analytic solutions. For example one could define a polynomial, trigonometric polynomial, or *pulse function*

$$U(\mathbf{x}, t) = (x^2 + 2xy + y^2 + z^2)(1 + \frac{1}{2}t + \frac{1}{3}t^2)$$
$$U(\mathbf{x}, t) = \cos(\pi\omega x)\cos(\pi\omega y)\cos(\pi\omega z)\cos(\omega_3 \pi t)$$
$$U(\mathbf{x}, t) = a_0 \exp(-a_1 \|\mathbf{x} - \mathbf{b}(t)\|^{2p}) \,, \quad \mathbf{b}(t) = \mathbf{c}_0 + \mathbf{v}t$$

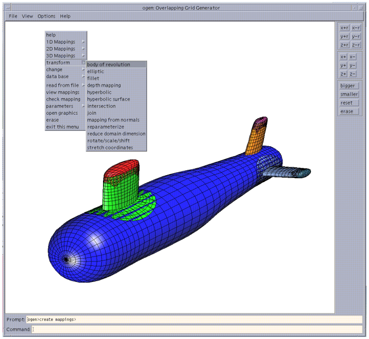
The polynomial solution is particularly useful since this solution is often an exact solution to the discrete equations on rectangular grids. The pulse function is good for AMR.

Grid Generation

Generating Overlapping Grids with Ogen...

The basic steps to follow when creating an overlapping grid are

1. create mappings that cover a domain and overlap where they meet.
2. generate the overlapping grid (the executable named ogen calls the grid generator Ogen).
3. save the grid in a data-base file.



```
ogen>create mappings>
```

A snapshot of the overlapping grid generator Ogen

Generating Overlapping Grids with Ogen...

1. **Ogen** determines the *holes* and interpolation information for an overlapping grid.
2. Input to **Ogen** is a set of overlapping grids, boundary conditions and shared-boundary information.
3. Options: *discretization width*, *interpolation width* are specified according to the intended equations and order of accuracy.
4. Boundaries are classified as physical boundaries, periodic or interpolation.
5. Physical boundaries are used to cut holes in overset grids.
6. The overlap is usually *minimized* to reduce the number of computational points.

Sample ogen command file for a square...

```
# make a grid for a square
create mappings
  rectangle
    mappingName
      square
    lines
      11 11
    boundary conditions
      1 1 1 1
  exit
exit
#
generate an overlapping grid
  square
  done
  change parameters
    ghost points
      all
      2 2 2 2 2 2
  exit
  compute overlap
exit
#
save an overlapping grid
  square10.hdf
  square10
exit
```

Note 1: A blank line denotes the **end** of the script!

Note 2: Comments using # only work starting with v24. A “$*$” can also be used.

Using Perl commands in Overture commands files.

Scripts can be parameterized. Command line arguments can also be read.

Perl is a powerful scripting language.

When an Overture program is started, a perl interpreter is created.

As command files are read by Overture:

1. Any line containing a semi-colon “;” is sent to the perl interpreter and discarded.
2. Any line with a “$” is evaluated by perl to replace variables.

Command line arguments can be passed to a command file. Ogen example:

```
ogen -noplot cicArg -order=4 -interp=e -factor=3
```

See Overture/sampleGrids/cicArg.cmd

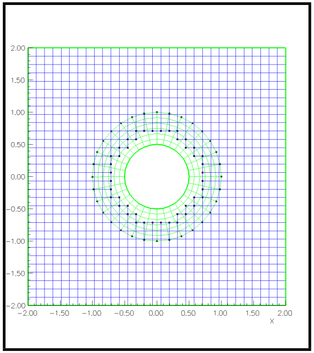
Using Perl commands in Overture commands files.

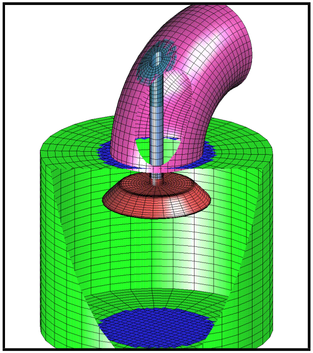
Here is a portion of an Ogen command file showing the use of perl commands

```
$factor=4;         # Define the grid resolution
$ds = .1/$factor;  # Target grid spacing
$pi = 4.*atan2(1.,1.);
...
Annulus
  $cx=.5; $cy=.75;                 # center of the annulus
  $innerRad=.5; $outerRad = .75;   # inner and outer radii
  center: $cx $cy
  inner and outer radii
    $innerRad $outerRad
  lines
    # compute the number of grid lines:
    $nr=int( ($outerRad-$innerRad)/$ds + 2.5 );
    $nTheta = int( 2.*$pi*($innerRad+$outerRad)*.5/$ds + 1.5 );
    $nTheta $nr
  boundary conditions
    -1 -1 5 0
  mappingName
   $annulusName
exit
```

Demo: Generating an Overlapping Grid with Ogen...

1. circle-in-a-channel example *by hand*.
2. Build a grid for a valve, port and cylinder using the native geometry tools. This example illustrates the building of a body-of-revolution and the creation of a special “join” mapping where the valve-stem intersects the port.





Constructing Grids on CAD Geometry

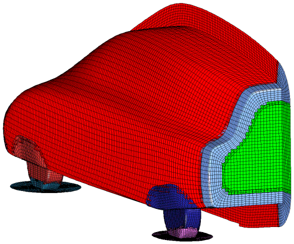
1. mBuilder and rap can be used to build grids for CAD geometries.
2. Surface grids are constructed first. Volume grids are generated starting from the surface grids.
3. Grids are usually built using the hyperbolic grid-generator hype.
4. Given a starting curve on the surface, a grid is generated by marching over the surface.
5. Volume grids are constructed by marching, starting from a surface grid.
6. Grids can be smoothed with an elliptic smoother. Grid lines can be clustered (stretched).

Demo: Constructing Grids on CAD geometries...

Building grids on the ASMO car geometry.

1. mbuilder asmoBody.cmd (grid generation for the car body).
2. mbuilder asmoFrontWheel.cmd (grid generation for the front wheel).
3. mbuilder asmoBackWheel.cmd (grid generation for the back wheel).
4. ogen asmo.cmd (overlapping grid generation).

(See Overture/sampleGrids/README.)



Primer...

Primer Examples: Using the High-Level Interface

Overture is distributed with a collection of *Primer* examples that can be used as a starting point for creating new solvers.

1. MappedGrid Examples: solving problems on single MappedGrid.
   - mappedGridExample2: Solve a convection diffusion equation on a single curvilinear grid.
2. Overlapping Grid Examples
   1. example 6: Solve a convection diffusion equation on an overlapping grid.
   2. example 7: Solve an elliptic boundary value problem.

mappedGridExample2: Solving $u_t + au_x + bu_y = \nu\Delta u$. Setup...

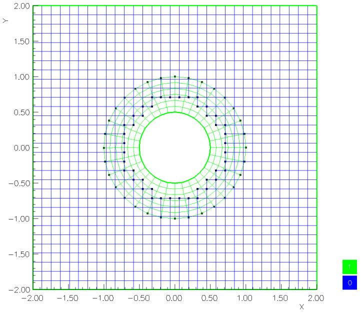
```
#include "Overture.h"
...
int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);              // initialize Overture
  AnnulusMapping annulus;
  MappedGrid mg(annulus);                  // MappedGrid for a square
  mg.update();                             // create default variables

  realMappedGridFunction u(mg);
  Index I1,I2,I3;
  getIndex(mg.dimension(),I1,I2,I3);       // assign I1,I2,I3 from dimension
  u(I1,I2,I3)=1.;                          // initial conditions

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

  PlotStuff ps(TRUE,"mappedGridExample2"); // create a PlotStuff object
  PlotStuffParameters psp;                 // This object is used to change plotting parameters
```

mappedGridExample2: $u_t + au_x + bu_y = \nu \Delta u$. Time-stepping...

```
real t=0, dt=.005, a=1., b=1., nu=.1;
for( int step=0; step<100; step++ )
{
  if( step \% 10 == 0 )
    PlotIt::contour(ps, u,psp );    // plot contours every 10 steps

  // ****** forward Euler time step *****
  u+=dt*( -a*u.x() -b*u.y() + nu*(u.xx()+u.yy()) );

  t+=dt;

  // apply Boundary conditions : u=0
  int component=0;
  u.applyBoundaryCondition(component,dirichlet,allBoundaries,0.);
  // fix up corners, periodic update:
  u.finishBoundaryConditions();
}

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

Example 6: Solving a PDE on an overlapping grid

Solve $u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$ on an overlapping grid.



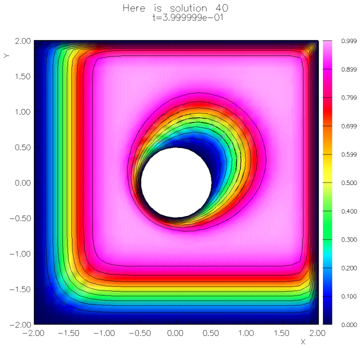


Figure: Results from example6.C, solve an advection-diffusion equation.

Example 6: Solving a PDE on an overlapping grid

Solve $u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$ on an overlapping grid.

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

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);   // initialize Overture
  aString nameOfOGFile="cic.hdf", nameOfShowFile="example6.show";

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

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

Example 6: Solving a PDE on an overlapping grid

continued...

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

  char buffer[80];                                // buffer for sprintf
  int numberOfTimeSteps=200;
  for( int i=0; i<numberOfTimeSteps; i++ )        // take some time steps
  {
    if( i % 40 == 0 )  // save solution every 40 steps
    {
      show.startFrame();                                    // start a new frame
      show.saveComment(0,sPrintF(buffer,"Here␣is␣solution␣%i",i));
      show.saveComment(1,sPrintF(buffer,"␣␣t=%e␣",t));
      show.saveSolution( u );
    }

    // *** take a time step with Euler's method ****
    u+=dt*( -a*u.x() - b*u.y() + viscosity*(u.xx() + u.yy()));

    t+=dt;
    u.interpolate();                                              // interpolate
    // apply a dirichlet BC on all boundaries:
    u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
    // for 4th order:
    // u.applyBoundaryCondition(0,BCTypes::extrapolate,BCTypes::allBoundaries,0.);
    u.finishBoundaryConditions();
  }

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

Example 7: Solve an elliptic boundary value problem

Solves $\Delta u + u_x = f$ with Dirichlet BC's on an overlapping grid.

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

int main(int argc, char *argv[])
{
  Overture::start(argc,argv);  // initialize Overture
  // create and read in a CompositeGrid
  aString nameOfOGFile="cic.hdf";
  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);
  cg.update();

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

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

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

Example 7: Solve an elliptic boundary value problem
continued...

```
  coeff=op.laplacianCoefficients()+op.xCoefficients();   // here is the operator
  // fill in the coefficients for the boundary conditions
  coeff.applyBoundaryConditionCoefficients(0,0,dirichlet,  allBoundaries);
  coeff.applyBoundaryConditionCoefficients(0,0,extrapolate,allBoundaries);
  coeff.finishBoundaryConditions();

  Oges solver( cg );                            // create a solver
  solver.setCoefficientArray( coeff );          // supply coefficients

  // assign the rhs: u=0 on the boundary
  Index I1,I2,I3, 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");
  Overture::finish();
  return(0);
}
```

Demo: Primer examples...

1. Solve a PDE on a mappedGrid:
    1. mappedGridExample2
2. Solve a PDE on an overlapping grid:
    1. example6
3. Deforming grid example:
    1. deform -numSteps=50

Other primer examples:

1. example9.C : read and write data to a DataBase file.
2. move1.C : moving grid example.
3. lins.C : solve the steady linearized Navier-Stokes equations.
4. callingFortran.C : how to call Fortran from C++.
5. pwave.C : a parallel wave equation solver.
6. gridPrint.C : read an Overture grid file (hdf) and output to a text file.
7. gridGenExample.C: build an overlapping grid in a program.
8. gridQuery.C : read an Oveture grid and display data.

AMR...

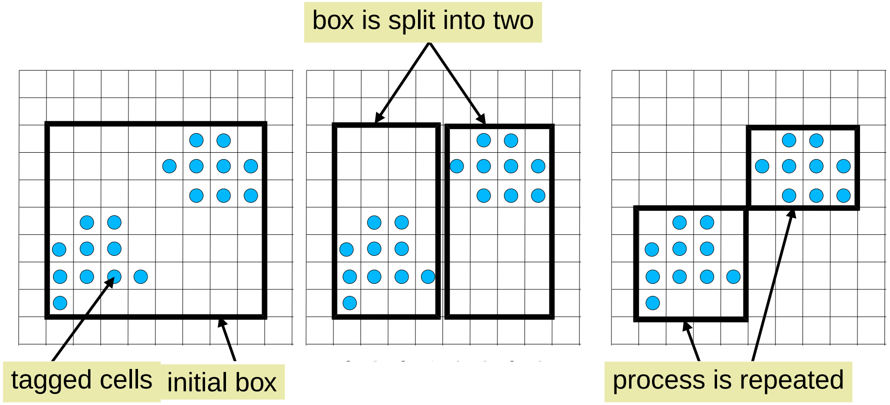
Block Structured Adaptive Mesh Refinement

1. Initially developed by Berger and Oliger (JCP 1984)
2. Extensions to the Euler equations by Berger and Colella (JCP 1989)
3. AMR and overlapping grids considered by Brislawn, Brown, Chesshire and Saltzman (1995), and Boden and Toro (1997)
4. AMR in Overture has contributions from Brown, Philip and Quinlan.

Some Structured AMR frameworks

1. AMRCLAW (LeVeque and Berger)
2. Amrita (Quirk)
3. AMROC (Deiterding, ORNL).
4. Boxlib (Bell et.al., LBNL)
5. Chombo (Colella et.al., LBNL)
6. GrACE (Parashar)
7. PARAMESH (NASA Goddard Space Flight Center)
8. SAMRAI (Hornung et.al. LLNL)

AMR regridding algorithm (Berger-Rigoutsos)



(1) tag cells where refinement is needed
(2) create a box to enclose tagged cells
(3) split box along longest direction (histogram of tagged cells)
(4) fit new boxes to each split box and repeat the steps as needed.

AMR on overlapping grids

Features of Overture's AMR:

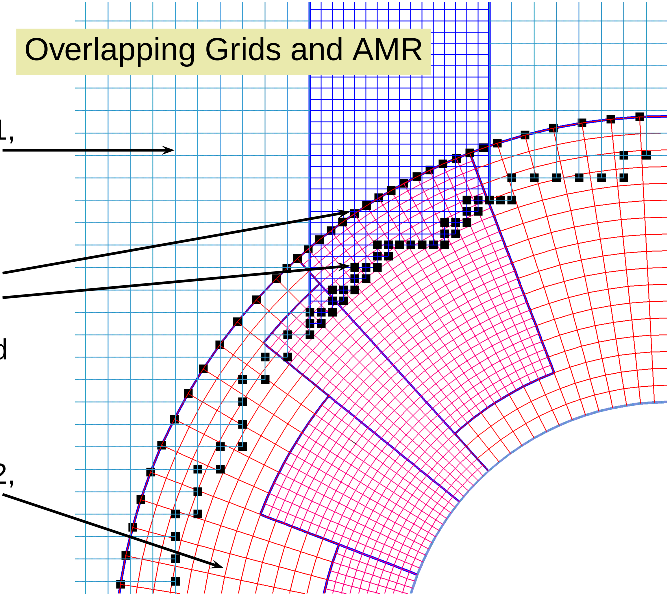
1. AMR grids are generated in the unit square coordinates of each component grid.
2. efficient handling of refinement grids on curvilinear grids.
3. parallel (although more work reqired to improve parallel scaling).
4. updating interpolation points between refinement grids from different base grids.
5. retaining the efficiency of Cartesian grids.
6. saving and reading solutions and grids from a data base file in an efficient manner (e.g. for post-processing and restarts).
7. interactive graphics.

Overlapping Grids and AMR

Component grid 1,
base grid 1

Refinement grids
interpolate from
refinements of a
different base grid

Component grid 2,
base grid 2



The basic AMR time stepping algorithm

```
PDEsolver( $\mathcal{G}, t_{\rm final}$ )
// $\mathcal{G}$ (input): current grid.
{
  t:=0; n:=0;
  $\mathbf{u}_{\mathbf{i}}^{n}$ := applyInitialCondition($\mathcal{G}$);
  while   $t < t_{\rm final}$
    if( n  mod  $n_{\rm regrid}$ == 0 )
      // regrid every $n_{\rm regrid}$ steps
      $e_{\mathbf{i}}$ := estimateError($\mathcal{G}, \mathbf{u}_{\mathbf{i}}^{n}$);
      $\mathcal{G}^*$ := regrid($\mathcal{G}, e_{\mathbf{i}}$);
      $\mathbf{u}_{\mathbf{i}}^{*}$ := interpolateToNewGrid($\mathbf{u}_{\mathbf{i}}^{n}, \mathcal{G}, \mathcal{G}^*$);
      $\mathcal{G} := \mathcal{G}^*$; $\mathbf{u}_{\mathbf{i}}^{n} := \mathbf{u}_{\mathbf{i}}^{*}$;
    $\Delta t$ := computeTimeStep($\mathcal{G}, \mathbf{u}_{\mathbf{i}}^{n}$);
    $\mathbf{u}_{\mathbf{i}}^{n+1}$ := timeStep($\mathcal{G}, \mathbf{u}_{\mathbf{i}}^{n}, \Delta t$);
    $t := t + \Delta t$; $n := n + 1$;
    interpolate($\mathcal{G}, \mathbf{u}_{\mathbf{i}}^{n}$);
    applyBoundaryConditions($\mathcal{G}, \mathbf{u}_{\mathbf{i}}^{n}, t$);
}
```

AMR components of Overture

class Regrid

- generation of aligned AMR grids using the Berger-Rigoutsos algorithm.
- generation of rotated AMR grids using the Berger algorithm.
- Boxlib is used for domain calculus (e.g. intersecting two boxes).

class ErrorEstimator

- defines standard error estimators based on first and second differences.
- smoothing of the error and propagation across overlapping grid boundaries.

class Interpolate

- fine to coarse and coarse to fine interpolation of patches.
- supports *any* refinement ratio (1,2,3,4,...) and *any* order of accuracy.

AMR components of Overture

class InterpolateRefinements

- high level function to interpolate the solution from one AMR overlapping grid to another AMR overlapping grid.
- update all AMR ghost points and hidden coarse grid points on an AMR overlapping grid.

Ogen

- the overlapping grid generator knows how to update interpolation points on AMR grids.

Elliptic Solvers

- Oges can be used to solve for the solution on the entire AMR hierarchy. Oges is an interface to sparse solvers such as PETSc.

amrh.C: a simple Overture AMR code

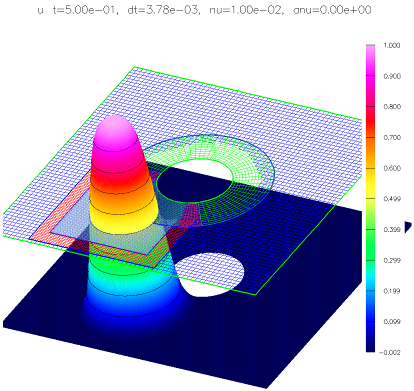
Found in Overture/primer/amrh.C

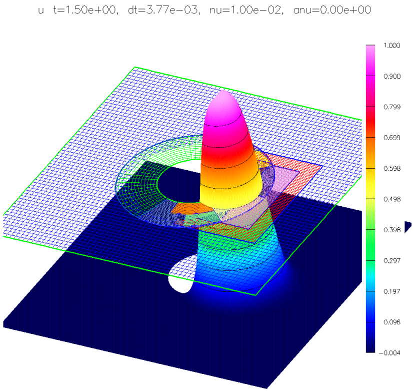
1. A small code demonstrating the use of AMR with Overture.
2. Solves the advection diffusion equation.
3. Runge-Kutta time-stepping
4. uses the *method of analytic solutions* for testing accuracy.

Example uses of Overture AMR functions from amrh.C:

```
CompositeGrid cg,cgNew;
realCompositeGridFunction u,error,uNew;

ErrorEstimator errorEstimator;
errorEstimator.computeAndSmoothErrorFunction(u,error);   // compute error estimate

Regrid regrid;
regrid.regrid(cg,cgNew, error, errorThreshold );         // generate new AMR grid

Ogen ogen;
ogen.updateRefinement(cgNew);        // compute AMR overlapping interp. pts.

uNew.updateToMatchGrid(cgNew);
InterpolateRefinements interp;
interp.interpolateRefinements( u,uNew );  // interpolate refinement grids
```

Example amrh computation





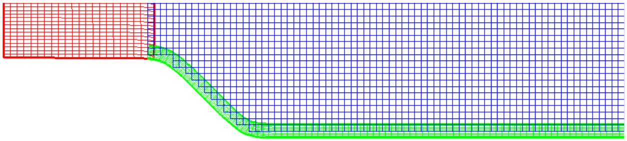
Traveling pulse analytic solution

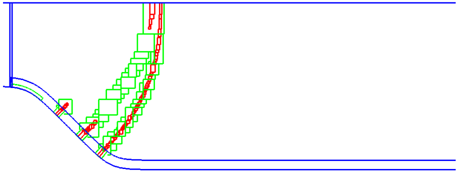
amrh demo

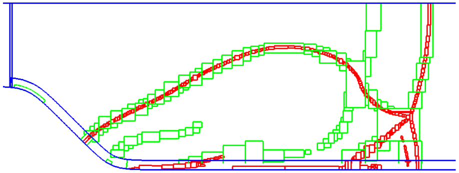
1. Propagation of a pulse through an overlapping grid. In this example the exact solution is a polynomial (twilightzone flow) but the error and AMR grids are based on a propagating pulse. The computed errors should be “zero” since the scheme should be exact for polynomials of degree 2.
    1. ogen -noplot sisArg.cmd -interp=e -factor=2
    2. amrh -cmd=amrhtz -g=sise2.order2.hdf -xc=-.4 -yc=-.4 -tf=1. -l=2 -r=2 -go=halt
    3. amrh -cmd=amrhtz -g=sise2.order2.hdf -xc=-.4 -yc=-.4 -tf=1. -l=3 -r=2 -go=halt

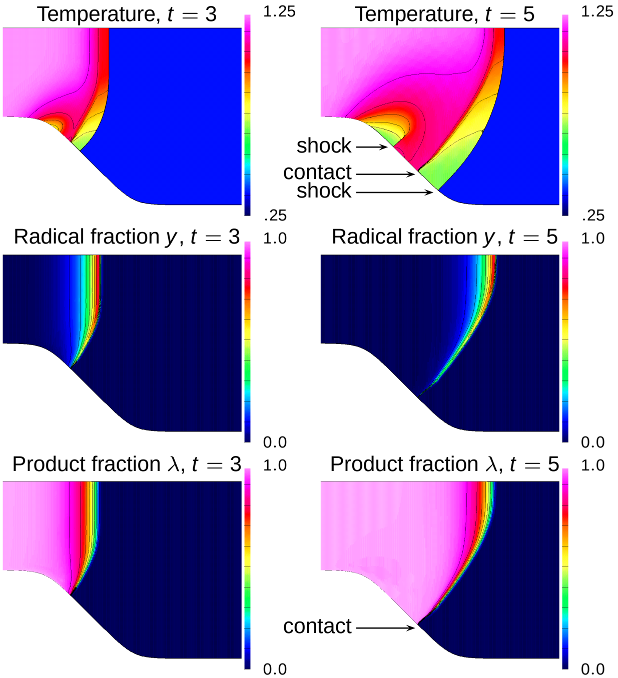
Detonation in an Expanding Channel

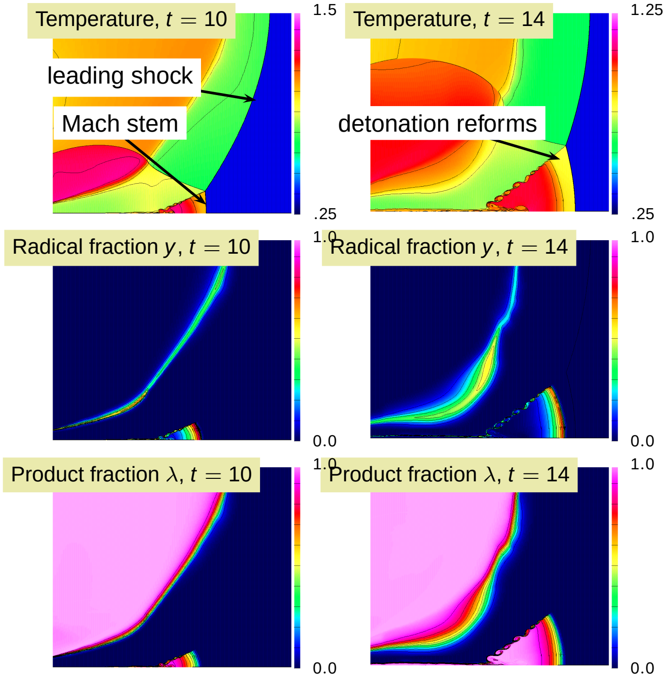
An example AMR computation using cgcns.








Temperature, $t=3$
1.25
.25
Temperature, $t=5$
1.25
.25
shock
contact
shock
Radical fraction y, $t=3$
1.0
0.0
Radical fraction y, $t=5$
1.0
0.0
Product fraction λ, $t=3$
1.0
0.0
Product fraction λ, $t=5$
1.0
0.0
contact


Temperature, $t = 10$
leading shock
Mach stem
1.5
.25
Temperature, $t = 14$
detonation reforms
1.25
.25
Radical fraction y, $t = 10$
1.0
0.0
Radical fraction y, $t = 14$
1.0
0.0
Product fraction λ, $t = 10$
1.0
0.0
Product fraction λ, $t = 14$
1.0
0.0

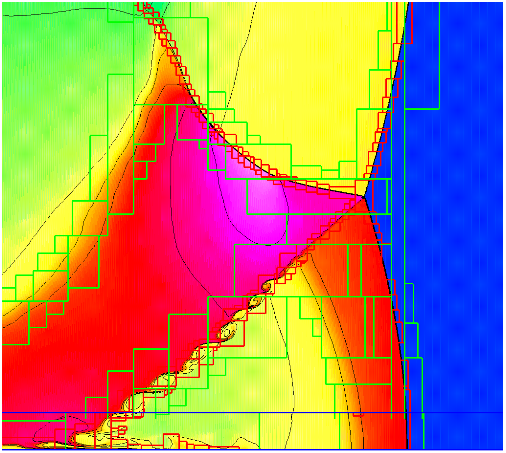


Figure: Closeup of the density near the Mach stem. The boundaries of the refinement grids are shown.

AMR (serial) performance on two problems:

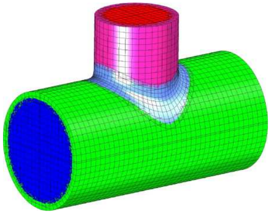
	Quarter plane		Expanding channel	
time steps	$12,418$		$21,030$	
grids (min,ave,max)	$(2,57,353)$		$(5,274,588)$	
points (min,ave,max)	$(2.0e5, 9.2e5, 1.9e6)$		$(1.2e5, 6.4e5, 1.3e6)$	
	s/step	%	s/step	%
compute $\Delta u_{i,j}^n$	13.85	92.7	11.50	82.4
boundary conditions	.12	.8	.14	1.0
interpolation (overlapping)	.09	.6	.45	3.2
AMR regrid/interpolation	.54	3.6	1.62	11.6
other	.34	2.3	.25	1.8
total	14.94	100	13.96	100

Table: CPU time (in seconds) per step for various parts of the code and their percentage of the total CPU time per step.

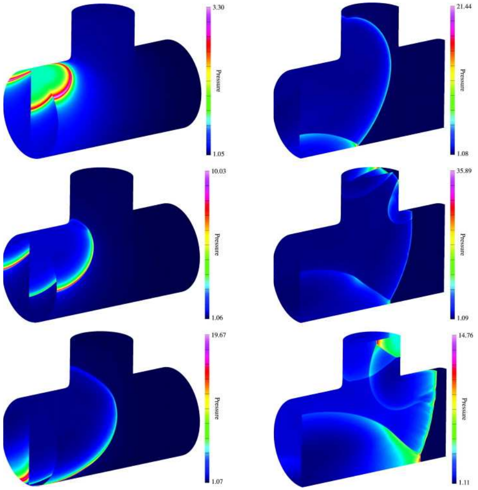
From

- WDH, DWS, *An Adaptive Numerical Scheme for High-Speed Reactive Flow on Overlapping Grids*, J. Comput. Phys., 2003.

Detonation initiation in a T-shaped pipe


overlapping grid

pressure



Notes: cgcns, reactive-Euler: one refinement level, factor 4, 4930 time steps, 48 processors, from 5 to 682 grids, 100M pts (max) (eff. resolution 400 M).

Estimating Convergence Rates

Define the volume-weighted discrete L_p-norm of a grid function $U_{\mathbf{i}}$ as

$$\|U_{\mathbf{i}}\|_p = \left(\frac{\sum_{\mathbf{i}} |U_{\mathbf{i}}|^p \, d\mathcal{V}_{\mathbf{i}}}{\sum_{\mathbf{i}} d\mathcal{V}_{\mathbf{i}}} \right)^{1/p}, \qquad d\mathcal{V}_{\mathbf{i}} = \left| \frac{\partial \mathbf{x}}{\partial \mathbf{r}} \right|_{\mathbf{i}} dr_1 dr_2 dr_3.$$

Assume the discrete solution $U_{\mathbf{i}}^m$ at grid spacing h_m satisfies

$$U_{\mathbf{i}}^m - u(\mathbf{x}_{\mathbf{i}}^m, t) \approx C_{\mathbf{i}}^m h_m^{\mu},$$

The difference between resolution h_n and h_m is

$$\|U_{\mathbf{i}}^m - \mathcal{R}_n^m U_{\mathbf{i}}^n\|_p \approx C |h_m^{\mu} - h_n^{\mu}|,$$

where $\mathcal{R}_n^m$ is a fine to coarse restriction operator.

Result: Given three solutions we can estimate the convergence rate μ and the error.

Estimating Convergence Rates

Detonation in a T-Pipe				
	$t = 2.0$		$t = 2.8$	
h_m	$\mathcal{E}_1^m$	$\mathcal{E}_2^m$	$\mathcal{E}_1^m$	$\mathcal{E}_2^m$
$1/120$	4.0e$-$3	3.0e$-$2	3.8e$-$2	2.6e$-$1
$1/160$	2.2e$-$3	1.6e$-$2	2.4e$-$2	1.9e$-$1
$1/240$	9.8e$-$4	7.1e$-$3	1.2e$-$2	1.2e$-$1
rate, μ	2.04	2.07	1.65	1.09

Estimated L_1 and L_2 errors in the density, $\mathcal{E}_1^m$ and $\mathcal{E}_2^m$, respectively, and convergence rates μ at $t = 2.0$ and $t = 2.8$.

- WDH, DWS, *Parallel Computation of Three-Dimensional Flows using Overlapping Grids with Adaptive Mesh Refinement*, J. Comput. Phys., 2008.

PDE Solvers...

An Overview of Overture based PDE Solvers

1. Oges: equation solver for implicit systems.
2. Ogmg: fast multigrid solver for scalar elliptic equations.
3. cgad : advection-diffusion solver.
4. cgins : incompressible flow.
5. cgcns : compressible flow with adaptive mesh refinement.
6. cgmx : Maxwell’s equations.
7. cgsm : Elastic wave equation (*new*).
8. cgmp : multi-domain multi-physics problems.
    1. conjugate heat transfer (fluid flow and solid heat transfer).
    2. fluid-structure interactions (FSI) (*under development*).

Oges: Overlapping Grid Equation Solver

The Oges class can be used to solve the systems of sparse equations that result from discretising boundary value problems (elliptic problems, implicit time-stepping) on overlapping grids.
Oges is an interface to various solvers packages:

1. sparse direct solvers (e.g. Yale, SuperLU)
2. sparse iterative solvers (SLAP, PETSc)
3. multigrid solver (Ogmg)

Primer example7 shows the use of Oges

Here the sparse matrix for the operator $\text{coeff} = \Delta + \partial_x$ is formed.

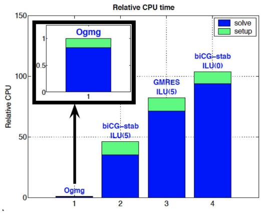
```
...
realCompositeGridFunction  coeff(cg,stencilSize,all,all,all);
CompositeGridOperators op(cg);            // create some differential operators
...
coeff=op.laplacianCoefficients()+op.xCoefficients();    // here is the operator
...
Oges solver( cg );                       // create an Oges solver
solver.setCoefficientArray( coeff );     // supply coefficients
...
solver.solve( u,f );
```

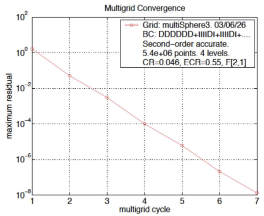
Ogmg is a multigrid solver for overlapping grids.

Currently being extended to 4th-order accuracy and parallel.

Ogmg is many times faster than Krylov methods.

- matrix-free; optimized for Cartesian grids.
- automatic coarse grid generation.
- adaptive smoothing
  - variable sub-smooths per component grid.
  - interpolation-boundary smoothing (IBS).
- Galerkin coarse grid operators (operator averaging).
- *centered* numerical boundary conditions for Dirichlet and Neumann problems.



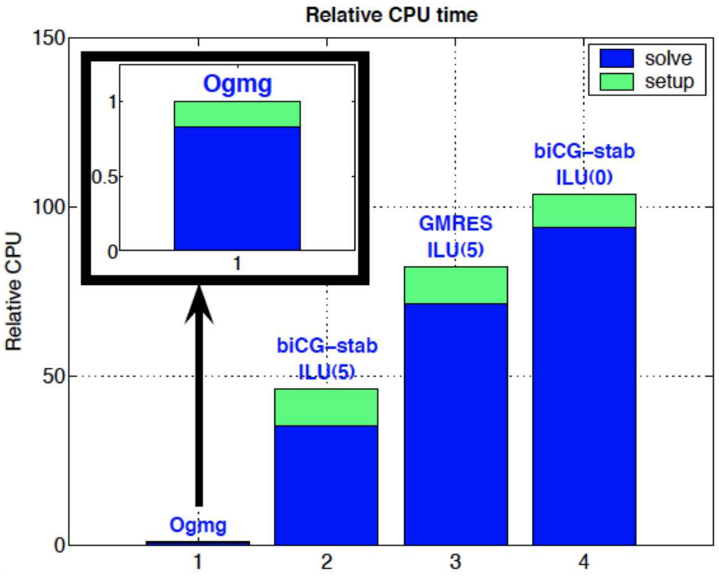


WDH., *On Multigrid For Overlapping Grids*, SIAM J. Sci. Comput. **26**, no. 5, (2005) 1547–1572.

Next

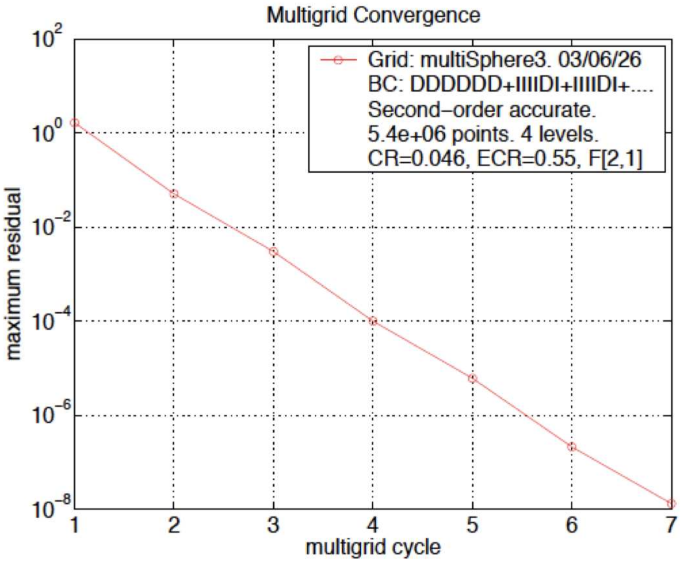
Ogmg is a multigrid solver for overlapping grids.

Currently being extended to 4th-order accuracy and parallel.

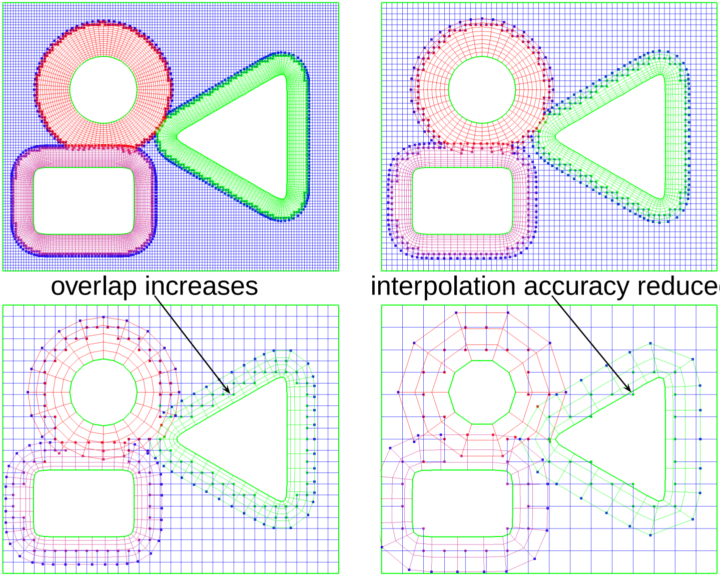


Ogmg is a multigrid solver for overlapping grids.

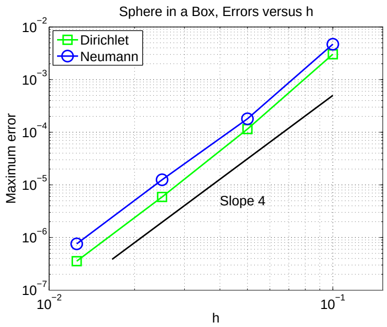
Currently being extended to 4th-order accuracy and parallel.

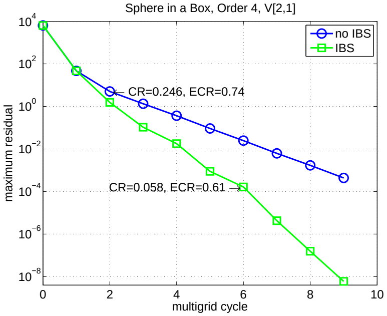


Automatic coarse grid generation is a key feature.



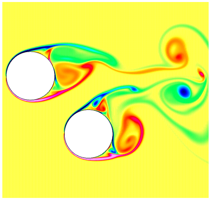
Accuracy and convergence of the new fourth-order accurate parallel version of Ogmg.

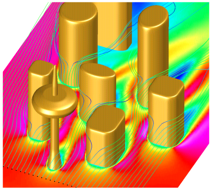




- Initial results indictate that the fourth order multigrid solver will be orders of magnitude faster than competing solvers.

Cgins: incompressible Navier-Stokes solver.





- efficient split-step scheme.
- spatial approximations: 2nd-order and 4th-order accurate.
- time-stepping methods: explicit, semi-implicit (time accurate), pseudo steady-state (efficient line solver), full implicit.
- New: approximate factored scheme with high-order compact approximations.
- support for moving rigid-bodies.
- heat transfer (Boussinesq approximation).

• WDH., *A Fourth-Order Accurate Method for the Incompressible Navier-Stokes Equations on Overlapping Grids*, J. Comput. Phys, **113**, no. 1, (1994) 13–25.
• WDH, N.A. Petersson, *A Split-Step Scheme for the Incompressible Navier-Stokes Equations*, in *Numerical Simulation of Incompressible Flows*, 2003.

Cgins solves the velocity-pressure form of the Navier-Stokes equations.

Incompressible Navier-Stokes equations: velocity-divergence form:

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, & t>0, \quad \mathbf{x}\in\Omega \\ \nabla\cdot\mathbf{u} &= 0 & t>0, \quad \mathbf{x}\in\Omega \end{aligned}$$

Cgins solves the velocity pressure form:

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, & t>0, \quad \mathbf{x}\in\Omega \\ \Delta p - \nabla\mathbf{u} : \nabla\mathbf{u} - \nabla\cdot\mathbf{f} &= 0, & t>0, \quad \mathbf{x}\in\Omega \end{aligned}$$

The split-step scheme decouples the solution of the velocity and pressure to avoid solving a large coupled system of equations.

The high-order accurate split-step scheme relies on careful treatment of the boundary conditions.

Split-step schemes are more efficient than coupled schemes but many split-step schemes are only first-order accurate.

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, && t>0, \quad \mathbf{x}\in\Omega \\ \Delta p - \nabla\mathbf{u} : \nabla\mathbf{u} - \alpha\nabla\cdot\mathbf{u} - \nabla\cdot\mathbf{f} &= 0, && t>0, \quad \mathbf{x}\in\Omega \end{aligned}$$

Divergence damping term: $\alpha\nabla\cdot\mathbf{u}$ is important.

Wall boundary conditions:

$$\mathbf{u} = 0, \quad \nabla\cdot\mathbf{u} = 0, \ \text{(pressure BC)} \quad \mathbf{x}\in\partial\Omega,$$

with *numerical boundary condition*:

$$p_n = -\mathbf{n}\cdot(\,\nu\nabla\times\nabla\times\mathbf{u}\,).$$

Use $\nabla\times\nabla\times\mathbf{u}$ instead of $\Delta\mathbf{u}$ for implicit time-stepping.

The high-order accurate split-step scheme relies on careful treatment of the boundary conditions.

Split-step schemes are more efficient than coupled schemes but many split-step schemes are only first-order accurate.

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, & t > 0, \quad \mathbf{x}\in\Omega \\ \Delta p - \nabla\mathbf{u} : \nabla\mathbf{u} - \alpha\nabla\cdot\mathbf{u} - \nabla\cdot\mathbf{f} &= 0, & t > 0, \quad \mathbf{x}\in\Omega \end{aligned}$$

Divergence damping term: $\alpha\nabla\cdot\mathbf{u}$ is important.

Wall boundary conditions:

$$\mathbf{u} = 0, \quad \nabla\cdot\mathbf{u} = 0, \text{ (pressure BC)} \quad \mathbf{x}\in\partial\Omega,$$

with *numerical boundary condition*:

$$p_n = -\mathbf{n}\cdot(\,\nu\nabla\times\nabla\times\mathbf{u}\,).$$

Use $\nabla\times\nabla\times\mathbf{u}$ instead of $\Delta\mathbf{u}$ for implicit time-stepping.

The high-order accurate split-step scheme relies on careful treatment of the boundary conditions.

Split-step schemes are more efficient than coupled schemes but many split-step schemes are only first-order accurate.

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, \qquad t>0, \quad \mathbf{x}\in\Omega \\ \Delta p - \nabla\mathbf{u} : \nabla\mathbf{u} - \alpha\nabla\cdot\mathbf{u} - \nabla\cdot\mathbf{f} &= 0, \qquad t>0, \quad \mathbf{x}\in\Omega \end{aligned}$$

Divergence damping term: $\alpha\nabla\cdot\mathbf{u}$ is important.

Wall boundary conditions:

$$\mathbf{u} = 0, \quad \nabla\cdot\mathbf{u} = 0, \ \text{(pressure BC)} \quad \mathbf{x}\in\partial\Omega,$$

with *numerical boundary condition*:

$$p_n = -\mathbf{n}\cdot(\,\nu\nabla\times\nabla\times\mathbf{u}\,).$$

Use $\nabla\times\nabla\times\mathbf{u}$ instead of $\Delta\mathbf{u}$ for implicit time-stepping.

The high-order accurate split-step scheme relies on careful treatment of the boundary conditions.

Split-step schemes are more efficient than coupled schemes but many split-step schemes are only first-order accurate.

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, \qquad t>0, \quad \mathbf{x}\in\Omega \\ \Delta p - \nabla\mathbf{u} : \nabla\mathbf{u} - \alpha\nabla\cdot\mathbf{u} - \nabla\cdot\mathbf{f} &= 0, \qquad t>0, \quad \mathbf{x}\in\Omega \end{aligned}$$

Divergence damping term: $\alpha\nabla\cdot\mathbf{u}$ is important.

Wall boundary conditions:

$$\mathbf{u} = 0, \quad \nabla\cdot\mathbf{u} = 0, \ \text{(pressure BC)} \quad \mathbf{x}\in\partial\Omega,$$

with *numerical boundary condition*:

$$p_n = -\mathbf{n}\cdot(\,\nu\nabla\times\nabla\times\mathbf{u}\,).$$

Use $\nabla\times\nabla\times\mathbf{u}$ instead of $\Delta\mathbf{u}$ for implicit time-stepping.

The high-order accurate split-step scheme relies on careful treatment of the boundary conditions.

Split-step schemes are more efficient than coupled schemes but many split-step schemes are only first-order accurate.

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \nu\Delta\mathbf{u} - \mathbf{f} &= 0, \qquad t>0, \quad \mathbf{x}\in\Omega \\ \Delta p - \nabla\mathbf{u} : \nabla\mathbf{u} - \alpha\nabla\cdot\mathbf{u} - \nabla\cdot\mathbf{f} &= 0, \qquad t>0, \quad \mathbf{x}\in\Omega \end{aligned}$$

Divergence damping term: $\alpha\nabla\cdot\mathbf{u}$ is important.

Wall boundary conditions:

$$\mathbf{u} \;=\; 0, \quad \nabla\cdot\mathbf{u} = 0, \text{ (pressure BC)} \quad \mathbf{x}\in\partial\Omega,$$

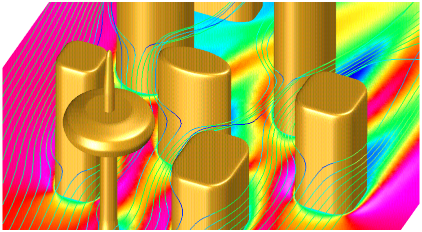
with *numerical boundary condition*:

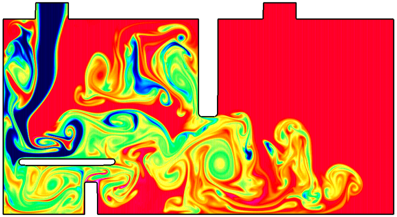
$$p_n \;=\; -\mathbf{n}\cdot(\,\nu\nabla\times\nabla\times\mathbf{u}\,).$$

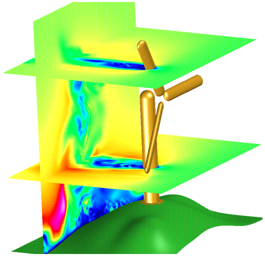
Use $\nabla\times\nabla\times\mathbf{u}$ instead of $\Delta\mathbf{u}$ for implicit time-stepping.

Incompressible flow computations with Cgins

Flow past 2d buildings

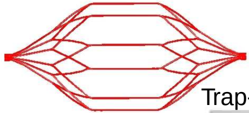

Flow past 3d buildings


Cool air entering a room (temperature)

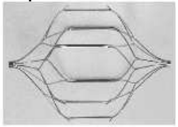

CgWind LES computation

Flow past a blood-clot filter using cgins

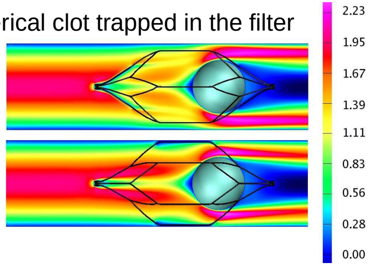
Overlapping grid for the filter



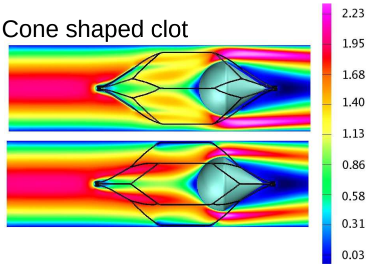
Trap-ease wire filter



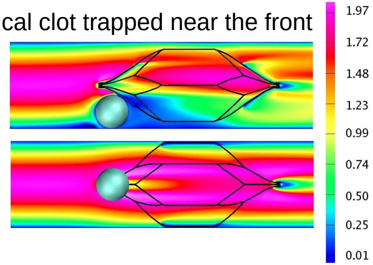
Spherical clot trapped in the filter



Cone shaped clot



Spherical clot trapped near the front



M.A. Singer, WDH, S.L. Wang, *Computational Modeling of Blood Flow in the Trapease Inferior Vena Cava Filter*, Journal of Vascular and Interventional Radiology, **20**, 2009.

Cgins Movies

tcilcVorticity.mpg high Re flow past two cylinders

joukowskyPitchPlunge.mpg pitching plunging airfoil

drops-speed.mpg 5 cylinders falling in a channel

dropStick.mpg fluttering plate (light body).

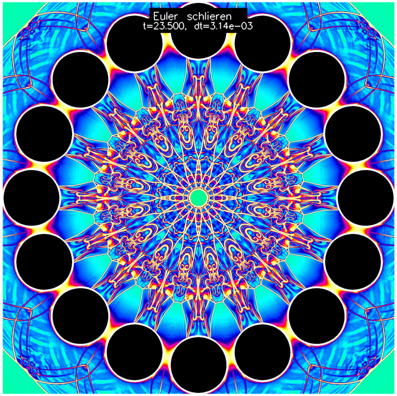
heatedRoom.mpg cool air flowing into a room.

sibDeform.mpg 3d deforming sphere

Cgins Demos

1. Flow past a cylinder.
   1. ogen -noplot cilcArg -interp=e -factor=1
   2. cgins cylinder -g=cilce1.order2.hdf -tf=100. -tp=1. -show="cyinder.show"
2. Two falling cylinders.
   1. ogen -noplot twoDropArg -interp=i -factor=2
   2. cgins twoDrop -g=twoDropi2.order2.hdf -nu=.02 -tf=7. -tp=.1
3. Heated cylinder in a square.
   1. ogen noplot cicArg -interp=e -factor=4
   2. cgins heatedCyl -g=cice4.order2 -nu=.01 -tf=10. -tp=1. -ts=implicit -solver=yale -psolver=yale

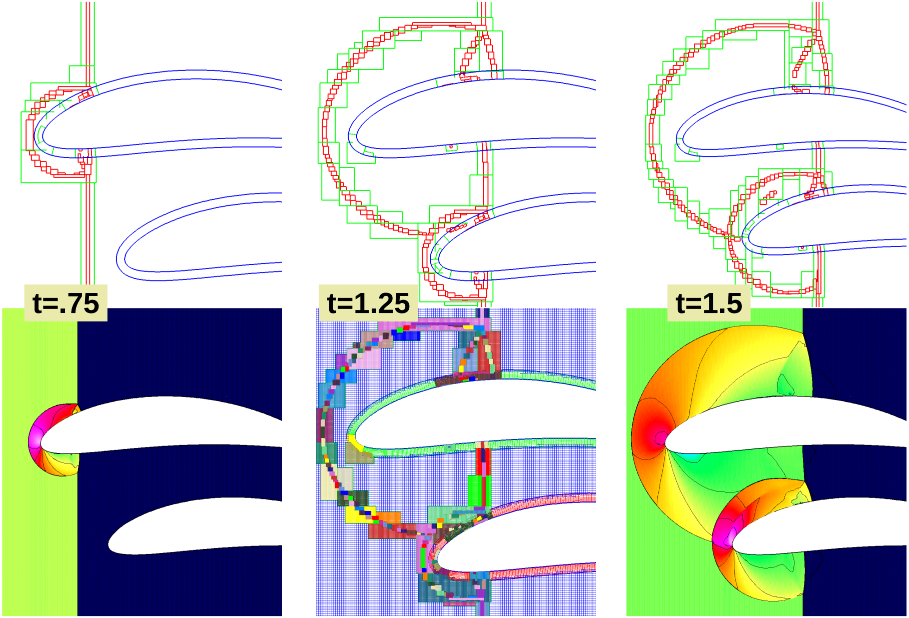
Cgcns: compressible N-S and reactive-Euler.



- reactive and non-reactive Euler equations, Don Schwendeman (RPI).
- compressible Navier-Stokes.
- multi-fluid formulation, Jeff Banks (LLNL).
- adaptive mesh refinement and moving grids.

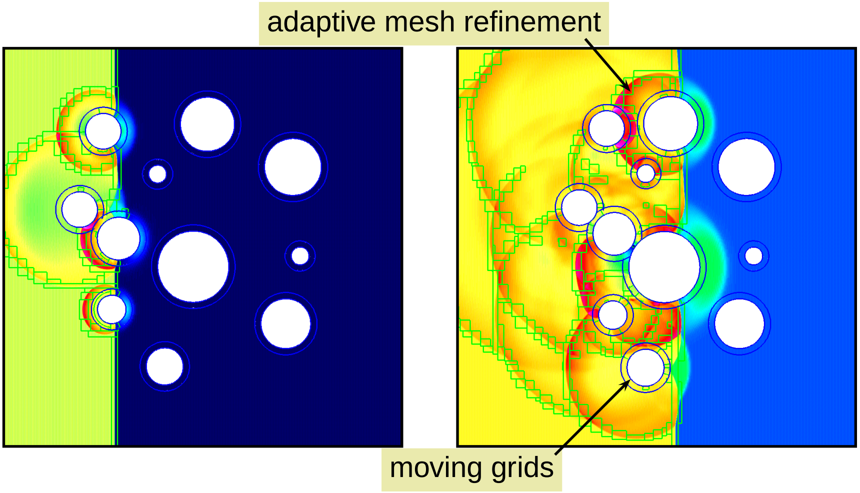
• WDH., D. W. Schwendeman, *Parallel Computation of Three-Dimensional Flows using Overlapping Grids with Adaptive Mesh Refinement*, J. Comp. Phys. **227** (2008).
• WDH., DWS, *Moving Overlapping Grids with Adaptive Mesh Refinement for High-Speed Reactive and Nonreactive Flow*, J. Comp. Phys. **216** (2005).
• WDH., DWS, An adaptive numerical scheme for high-speed reactive flow on overlapping grids, J. Comp. Phys. **191** (2003).

Cgcns supports adaptive mesh refinement



Moving overlapping grids and AMR

A shock hitting a collection of cylinders (density).



Cgcns Movies - High-speed Compressible Flow

randomCylSchlieren.mpg : shock hitting multiple cylinders

randomCylGrids.mpg : AMR grids for shock hitting multiple cylinders

movingPlate78-8big.mpg : very high speed plate hitting many particles.

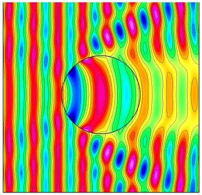
detCell-L3-Rho.mpg : cellular detonation

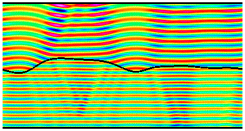
detTube.mpg : 3D perturbed detonation in a tube.

Demo: Cgcns - High-speed Compressible Flow

1. Shock hitting a cylinder (with AMR).
    1. ogen noplot cicArg -interp=e -factor=2
    2. cgcns cicShockg.cmd -g=cice2.order2.hdf -l=2 -r=2 -tf=1. -tp=.1 -xStep="x=-1.5"
2. Shock hitting a moving cyinder (with and without AMR).
    1. cgcns cicShockMove -g=cice2.order2
    2. cgcns cicShockMove -g=cice2.order2 -amr=1

Cgmx: electromagnetics solver.





- a time-domain finite difference scheme.
- fourth-order accurate, 2D, 3D.
- Efficient time-stepping with the modified-equation approach
- High-order accurate symmetric difference approximations.
- High-order-accurate *centered* boundary and interface conditions.

• WDH., *A High-Order Accurate Parallel Solver for Maxwell's Equations on Overlapping Grids*, SIAM J. Scientific Computing, **28**, no. 5, (2006).

Maxwell's equations are solved in second-order form

Maxwell's equations:

$$\epsilon\mu\ \partial_t^2\mathbf{E} = \Delta\mathbf{E} + \nabla\Big(\nabla\ln\epsilon\ \cdot\mathbf{E}\Big) + \nabla\ln\mu\times\Big(\nabla\times\mathbf{E}\Big) - \mu\partial_t\mathbf{j}$$

$$\epsilon\mu\ \partial_t^2\mathbf{H} = \Delta\mathbf{H} + \nabla\Big(\nabla\ln\mu\ \cdot\mathbf{H}\Big) + \nabla\ln\epsilon\times\Big(\nabla\times\mathbf{H}\Big) + \epsilon\nabla\times(\frac{1}{\epsilon}\mathbf{j})$$

Advantages of the second-order form:

- No need for a staggered grid since the operator Δ is elliptic.
- One can solve for **E** alone.

Maxwell's equations are solved in second-order form

Maxwell's equations:

$$\epsilon\mu\, \partial_t^2 \mathbf{E} = \Delta \mathbf{E} + \nabla\Big(\nabla \ln \epsilon \;\cdot \mathbf{E}\Big) + \nabla \ln \mu \times \Big(\nabla \times \mathbf{E}\Big) - \mu \partial_t \mathbf{j}$$

$$\epsilon\mu\, \partial_t^2 \mathbf{H} = \Delta \mathbf{H} + \nabla\Big(\nabla \ln \mu \;\cdot \mathbf{H}\Big) + \nabla \ln \epsilon \times \Big(\nabla \times \mathbf{H}\Big) + \epsilon \nabla \times (\frac{1}{\epsilon}\mathbf{j})$$

Advantages of the second-order form:

- No need for a staggered grid since the operator Δ is elliptic.
- One can solve for **E** alone.

Modified Equation time stepping

Taylor series in time:

$$\frac{u(t+\Delta t) - 2u(t) + u(t-\Delta t)}{\Delta t^2} = u_{tt} + \frac{\Delta t^2}{12} u_{tttt} + O(\Delta t^4)$$

For the wave equation

$$u_{tt} = \Delta u$$

a fourth-order scheme in space and time is

$$\frac{U_i^{n+1} - 2U_i^n + U_i^{n-1}}{\Delta t^2} = \Delta_{4h} U_i^n + \frac{\Delta t^2}{12} (\Delta^2)_{2h} U_i^n$$

This scheme is very efficient (especially on Cartesian grids) and allows a large (cfl=1) time step.

Modified Equation time stepping

Taylor series in time:

$$\frac{u(t+\Delta t) - 2u(t) + u(t-\Delta t)}{\Delta t^2} = u_{tt} + \frac{\Delta t^2}{12} u_{tttt} + O(\Delta t^4)$$

For the wave equation

$$u_{tt} = \Delta u$$

a fourth-order scheme in space and time is

$$\frac{U_i^{n+1} - 2U_i^n + U_i^{n-1}}{\Delta t^2} = \Delta_{4h} U_i^n + \frac{\Delta t^2}{12} (\Delta^2)_{2h} U_i^n$$

This scheme is very efficient (especially on Cartesian grids) and allows a large (cfl=1) time step.

Modified Equation time stepping

Taylor series in time:

$$\frac{u(t+\Delta t) - 2u(t) + u(t-\Delta t)}{\Delta t^2} = u_{tt} + \frac{\Delta t^2}{12} u_{tttt} + O(\Delta t^4)$$

For the wave equation

$$u_{tt} = \Delta u$$

a fourth-order scheme in space and time is

$$\frac{U_i^{n+1} - 2U_i^n + U_i^{n-1}}{\Delta t^2} = \Delta_{4h} U_i^n + \frac{\Delta t^2}{12} (\Delta^2)_{2h} U_i^n$$

This scheme is very efficient (especially on Cartesian grids) and allows a large (cfl=1) time step.

Modified Equation time stepping

Taylor series in time:

$$\frac{u(t+\Delta t) - 2u(t) + u(t-\Delta t)}{\Delta t^2} = u_{tt} + \frac{\Delta t^2}{12} u_{tttt} + O(\Delta t^4)$$

For the wave equation

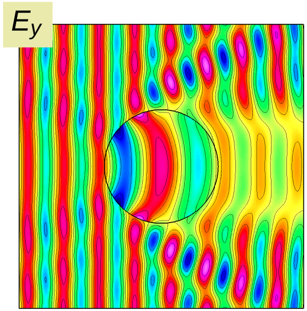
$$u_{tt} = \Delta u$$

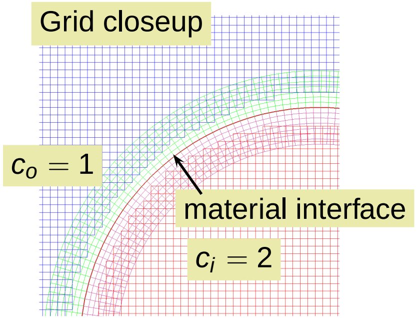
a fourth-order scheme in space and time is

$$\frac{U_i^{n+1} - 2U_i^n + U_i^{n-1}}{\Delta t^2} = \Delta_{4h} U_i^n + \frac{\Delta t^2}{12} (\Delta^2)_{2h} U_i^n$$

This scheme is very efficient (especially on Cartesian grids) and allows a large (cfl=1) time step.

Scattering of a plane wave by a dielectric cylinder

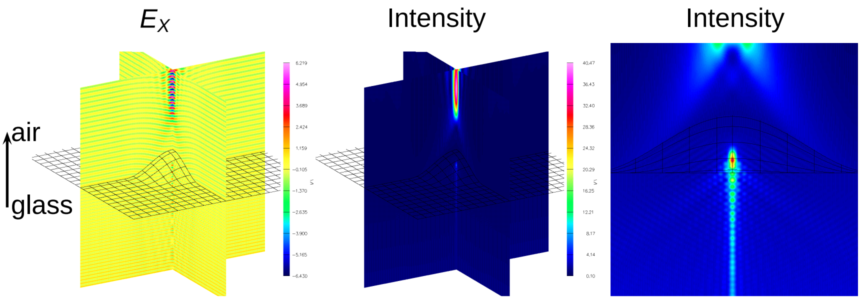




grid	$\lvert\lvert e^{E_x} \rvert\rvert_\infty$	$\lvert\lvert e^{E_y} \rvert\rvert_\infty$	$\lvert\lvert e^{H_z} \rvert\rvert_\infty$	$\delta_{\mathbf{E}}$
$\mathcal{G}_1$	$1.4e{-1}$	$2.9e{-1}$	$3.0e{-1}$	$6.7e{-2}$
$\mathcal{G}_2$	$1.0e{-2}$	$2.1e{-2}$	$2.2e{-2}$	$4.5e{-3}$
$\mathcal{G}_4$	$6.8e{-4}$	$1.4e{-3}$	$1.4e{-3}$	$2.9e{-4}$
rate σ	3.86	3.87	3.88	3.92

Known solution as a Mie series. Maximum errors at $t = 1$.

Scattering by a 3d material interface



- Uses newly developed 4th-order accurate 3D material interface approximations.
- Scattering of a plane wave by an interface with a bump, glass-to-air.
- 1 billion grid points, 32 nodes (8 processors per node) of a Linux cluster.

Cgmx Movies - Electromagnetic Wave Propagation

dieCyl16Ey.mpg Maxwell, dielectric cylinder (inside c=2)

dieCyl16Eps4Ey.mpg Maxwell, dielectric cylinder (inside c=.5)

afmOneky38.mp Maxwell, light travelling from glass to a vacuum

ilc2d.mpg Maxwell: wake fields in a 2D accelerator

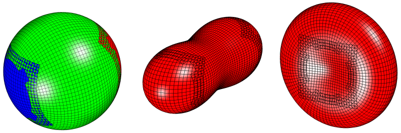
Demo: Cgmx - Electromagnetic Wave Propagation

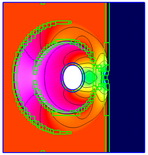
1. Diffraction by a dielectic cylinder.
    1. ogen noplot io -order=4 -interp=e -factor=4
    2. cgmx dielectricCyl -g=innerOutere4.order4 -kx=2 -eps1=.25 -eps2=1. -go=halt -dissOrder=4 -tp=.01 -tf=10
    3. cgmx dielectricCyl -g=innerOutere4.order4 -kx=2 -eps1=1. -eps2=.25 -go=halt -dissOrder=4 -tp=.01 -tf=10

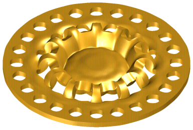
Cgsm: A solver for the elastic wave equation.

Available with Overture.v24

- for general 2D/3D overlapping grids, with dynamic AMR.
- Scheme SOS: finite difference scheme for the second-order-system.
- Scheme FOS: Godunov scheme for the first-order-system.
- efficient structured grid algorithms, optimized for Cartesian grids.
- stable traction BC's for the difficult case of $\lambda/\mu \gg 1$.


eigenmode of a solid sphere

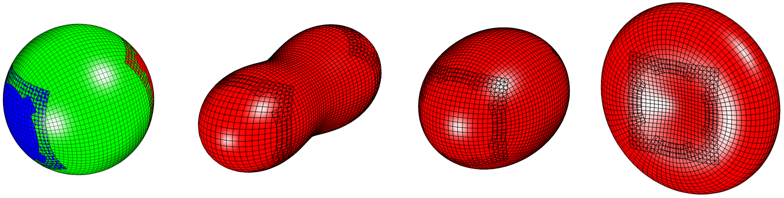

p-wave shock (AMR)


3D plate with holes

- D. Appelö, J.W. Banks , WDH, D.W. Schwendeman, *Numerical Methods for Solid Mechanics on Overlapping Grids: Linear Elasticity*, LLNL-JRNL-422223, submitted.

Cgsm: Vibrational mode of a solid elastic sphere.

Exact solution derived by Lamb, 1882.

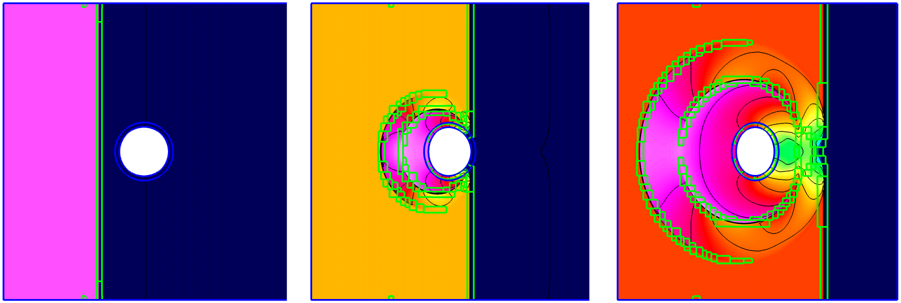


		SOS		FOS					
Grid $\mathcal{G}^{(j)}$	h_j	$e_u^{(j)}$	r	$e_u^{(j)}$	r	$e_v^{(j)}$	r	$e_\sigma^{(j)}$	r
$\mathcal{G}_{ss}^{(1)}$	1/10	$1.3e-1$		$5.1e-2$		$1.2e-1$		$2.6e-1$	
$\mathcal{G}_{ss}^{(2)}$	1/20	$4.0e-2$	3.2	$1.2e-2$	4.2	$3.0e-2$	4.0	$5.1e-2$	5.1
$\mathcal{G}_{ss}^{(4)}$	1/40	$10.0e-3$	4.0	$2.4e-3$	5.1	$7.1e-3$	4.2	$8.6e-3$	6.0
$\mathcal{G}_{ss}^{(8)}$	1/80	$2.4e-3$	4.1	$5.2e-4$	4.6	$1.7e-3$	4.1	$2.0e-3$	4.3
rate		1.93		2.22		2.03		2.37	

Maximum errors and estimated convergence rates.

Diffraction of a p-wave “shock” by a circular cavity.

With Adaptive Mesh Refinement.



Cgsm Movies - Elastic waves

plate3dWithHoles.mpg deforming 3D plate with holes.

deformingSphere.mpg deforming elastic sphere

Demo: Cgsm - Elastic waves

1. Deforming solid disk. (-pv=c : SOS, -pv=g : FOS).
   1. ogen noplot sicArg -factor=3 -order=2 -interp=e
   2. cgsm pulse -g=sice3.order2.hdf -pv=c -diss=1. -tp=.01 -tf=10. -x0=0. -y0=0.
   3. cgsm pulse -g=sice3.order2.hdf -pv=g -diss=0. -tp=.01 -tf=10. -x0=0. -y0=0.
2. Eigenmode of a 3D elastic solid sphere.
   1. ogen noplot sphere -interp=e -order=2 -factor=2
   2. cgsm sphereEigen -g=spheree2.order2 -diss=0.5 -tp=.05 -vClass=2 -nMode=2 -mMode=1 -go=halt -dsf=.05

Cgmp: a multi-domain multi-physics solver.

Cgmp:

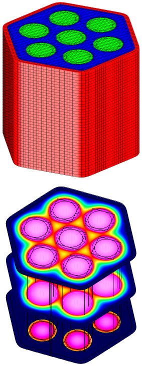
1. couples multiple CG solvers (cgcns, cgins, cgad, cgsm) into a single simulation.
2. a tightly-coupled partitioned approach for time-stepping (not a monolitic solver).

Examples:

1. Conjugate heat transfer (CHT): fluid flow (cgins) and heat transfer in solids (cgad).
2. Fluid structure interactions (FSI): fluid flow (cgcns) and solid defomation (cgsm) [under development].

Cgmp for conjugate heat transfer.

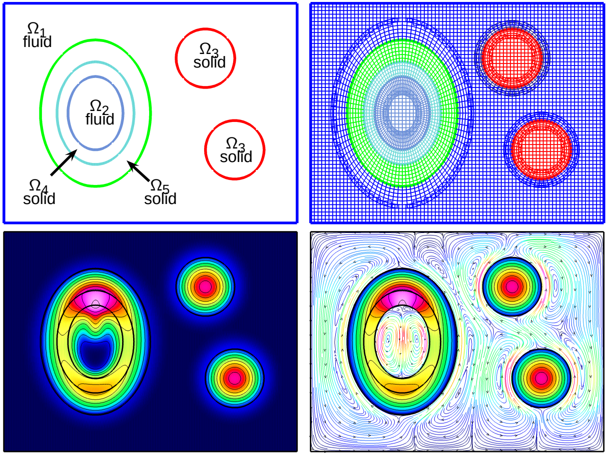
Coupling incompressible flow to heat conduction in solids.



- overlapping grids for each fluid or solid domain,
- a partitioned solution algorithm (separate physics solvers in each sub-domain),
- (cgins) incompressible Navier-Stokes equations (with Boussinesq approximation) for fluid domains,
- (cgad) heat equation for solid domains,
- a key issue is interface coupling.

- WDH., K. K. Chand, *A Composite Grid Solver for Conjugate Heat Transfer in Fluid-Structure Systems*, J. Comput. Phys, 2009.

The multi-domain composite grid approach for CHT



Each fluid or solid sub-domain is covered by an overlapping grid.
Fluid sub-domains : cgins. Solid sub-domains: cgad.
Coupled problem: cgmp.

Demo: Cgmp - Conjugate Heat Transfer

1. Four hot cylinders in a square.

   ```
   ogen noplot diskArray -factor=1 -interp=e -nCylx=2 -nCyly=2
   cgmp io.cmd -g="diskArray2x2ye1.order2.hdf" -nu=.05 -kappa=.01
   -tp=.05 -solver=yale
   ```

Deforming composite grids (DCG) for FSI

An efficient approach for FSI retaining high quality grids for large displacements.

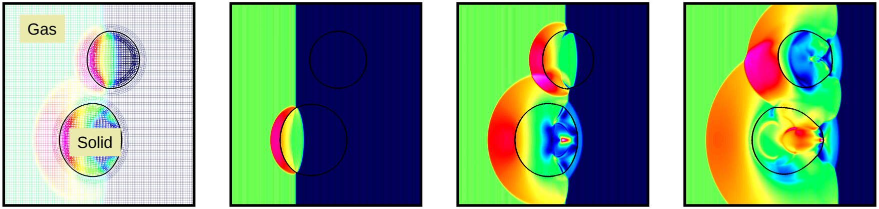
Goal: coupled simulations of compressible/incompressible fluids and deforming solids.

A mixed Eulerian-Lagrangian approach:

- Fluids: general moving coordinate system with overlapping grids.
- Solids : fixed reference frame with overlapping-grids (later: unstructured-grids, or beam/plate models).
- Boundary fitted deforming grids for fluid-solid interfaces.

Strengths of the approach:

- maintains high quality grids for large deformations/displacements.
- efficient structured grid methods (AMR) optimized for Cartesian grids.

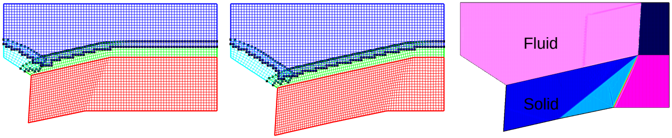


Mach 2 shock in a gas hitting two elastic cylinders.

Initial focus: coupling high speed compressible flow to linear elasticity.

1. interface approximations based on a fluid-solid Riemann problem are stable for all impedance ratios (e.g. the difficult case of *light solids*).
2. second-order accuracy demonstrated for problems with smooth solutions (elastic piston problem, deforming diffuser).
3. stable schemes for problem with shocks (e.g. superseismic shock).

- J.W. Banks, W.D. Henshaw, D.W. Schwendeman, *Deforming Composite Grids for Solving Fluid Structure Problems*, in preparation.
- J. W. Banks and B. Sjögreen, *A Normal Mode Stability Analysis of Numerical Interface Conditions for Fluid/Structure Interaction*, Commun. Comput. Phys., 2011.



Superseismic shock, grids and solution

Cgmp Movies - Fluid Structure Interactions

cavityDeform.mpg : converging shock in an elastic cavity

shockMultiDisk.mpg : shock hitting two elastic cylinders