Cgcns User Guide: An Overture Solver for the Compressible Navier–Stokes Equations on Composite Overlapping Grids,

William D. Henshaw,
Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy, NY, USA, 12180. June 26, 2021

Abstract:
Cgcns is a program that can be used to solve compressible fluid flow problems on overlapping grids. It is built upon the **Overture** object-oriented framework. Cgcns can be used to

- solve the compressible Navier–Stokes equations,
- solve the inviscid Euler equations with reactions,
- solve problems on moving grids,
- solve problems with adaptive mesh refinement.

Contents

1 Introduction

Cgcns is an compressible fluid flow solver for overlapping grids built upon the **Overture** framework [1],[3],[2]. More information about **Overture** can be found on the **Overture** home page, `http://www.llnl.gov/casc/Overture`.

Cgcns can be used to

- solve the compressible Navier–Stokes equations,
- solve the inviscid Euler equations with reactions,
- solve problems on moving grids,
- solve problems with adaptive mesh refinement.

The Cgcns solver is found in the `cns` directory in the **cg** distribution and has sub-directories

`bin` : contains the executable, cgcns. You may want to put this directory in your path.

`check` : contains regression tests.

`cmd` : sample command files for running cgcns, see section (2).

`lib` : contains the Cgcns library, `libCgcns.a`.

`src` : source files

Other documents of interest that are available through the **Overture** home page are

- The Cgcns Reference Guide [8] for detailed descriptions of the equations, algorithms and discretizations.
- The overlapping grid generator, `Ogen`, [5]. Use this program to make grids for cgcns.
- Mapping class documentation : `mapping.tex`, [4]. Many of the mappings that are used to create an overlapping grid are documented here.
- Interactive plotting : `PlotStuff.tex`, [7].
- `Oges` overlapping grid equation solver, used by cgcns to solve implicit time stepping equations and the Poisson equation for the pressure, [6].

1.1 Basic steps

Here are the basic steps to solve a problem with Cgcns.

1. Generate an overlapping grid with ogen. Make the grid with 2 ghost lines (this is the default).
2. Run cgcns (note lowercase 'c', found in the `bin/cgcns` directory) and choose the PDE you want to solve.
3. Assign the boundary conditions and initial conditions.
4. Choose the parameters for the PDE (viscosity, reaction rates, ...)
5. Choose run time parameters, time to integrate to, time stepping method etc.
6. Compute the solution (optionally plotting the results as the code runs).
7. When the code is finished you can look at the results (provided you saved a 'show file') using `plotStuff`.

The commands that you enter to run cgcns can be saved in a command file (by default they are saved in the file 'cgcns.cmd'). This command file can be used to re-run the same problem by typing 'cgcns file.cmd'. The command file can be edited to change parameters.

To get started you can run one of the demo's that come with cgcns, these are explained next in section (2).

Papers that describe some of the results obtained with Cgcns include [11, 10, 9].

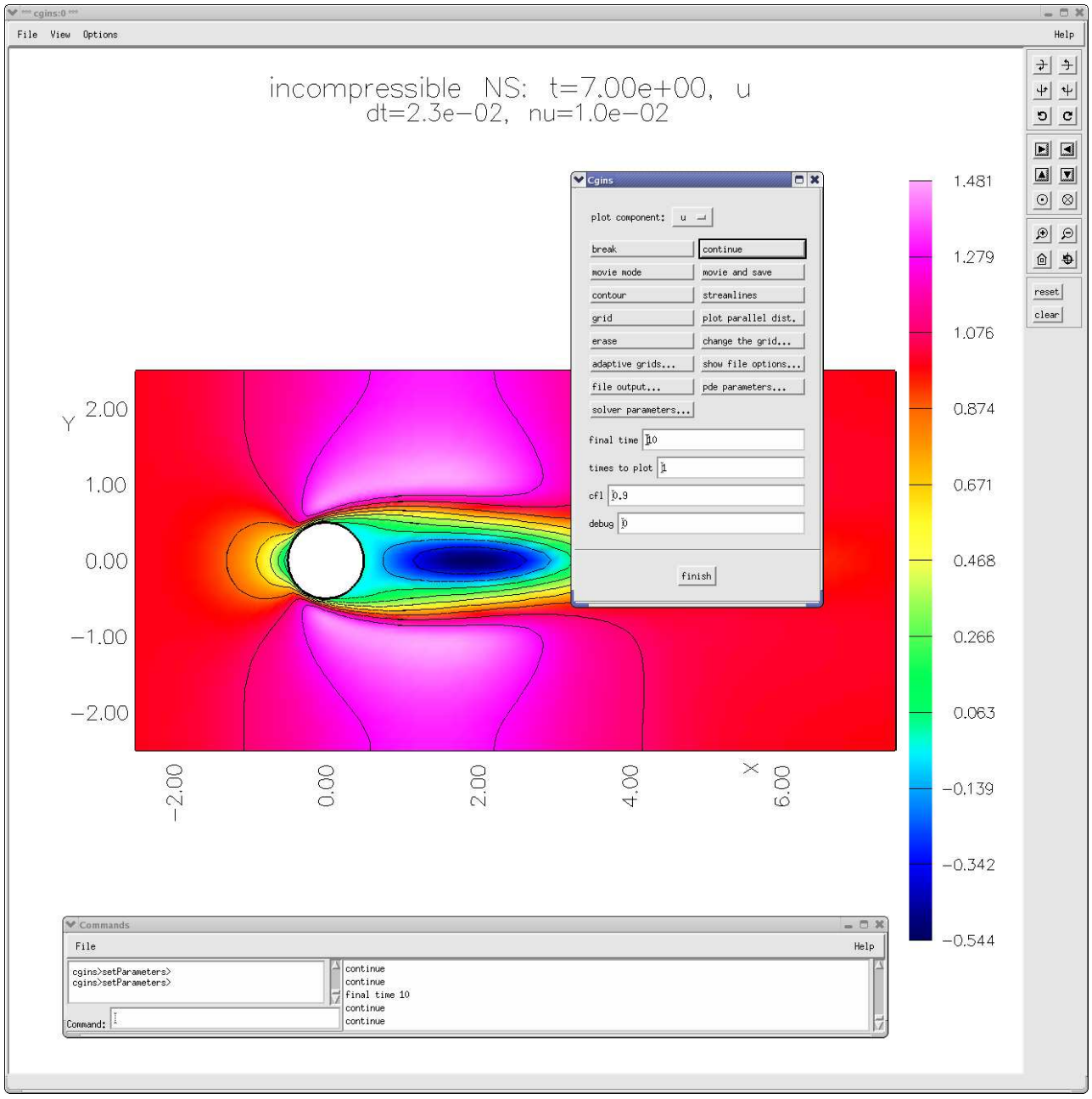

Figure 1: Snapshot of cgins (cgcns is similar) showing the run time dialog menu.

1.2 Specifying the boundary conditions correctly

It can be confusing to get all the boundary conditions correct. To help you do this you should plot the grid and display the boundaries coloured by the boundary condition number (this is the default) as shown in figure (2). In 3D you will need to `'plot shaded surfaces'` to see the boundary colours. This will help you see if all the faces are correct. Cgins prints out the number that corresponds to each boundary.

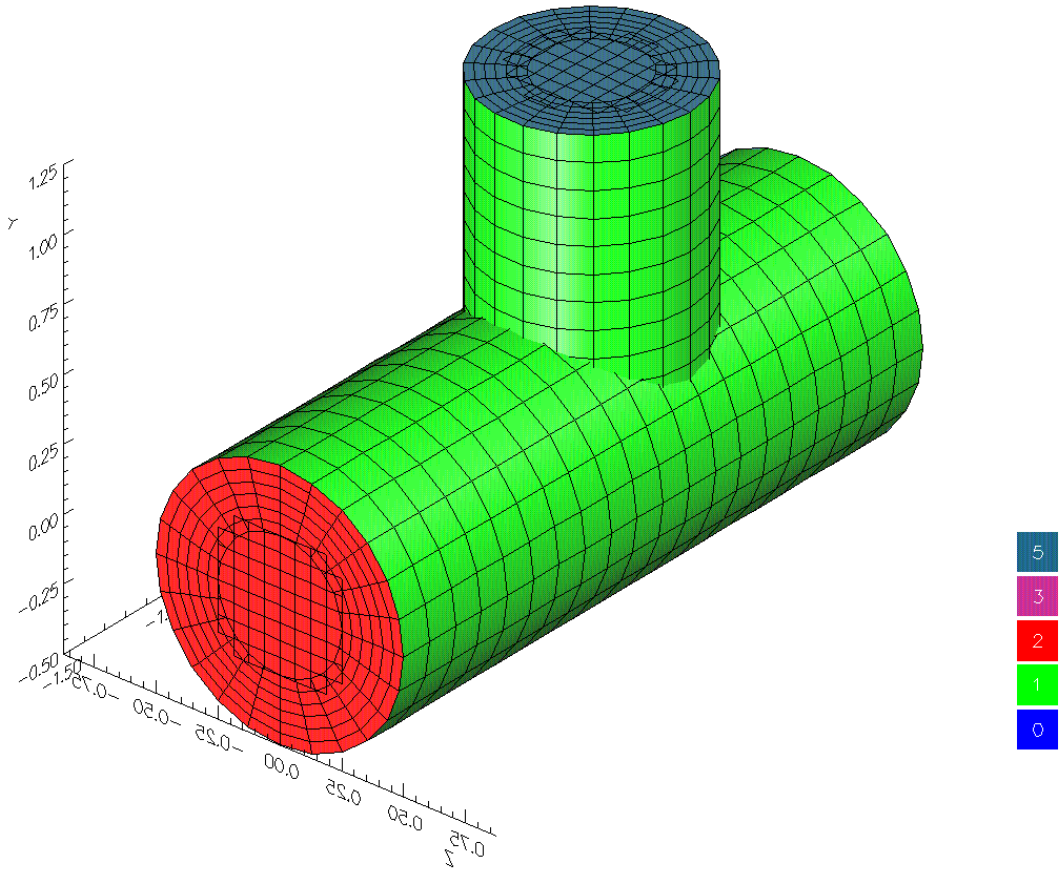

Figure 2: After specifying boundary conditions it is helpful to plot the grid with boundaries coloured by the boundary condition number. Here we see that the inflow boundary for the main pipe is number 2 (`inflowWithVelocityGiven`) the outflow boundary for the branch pipe is number 5 (`outflow`) and the walls are number 1 (`noSlipWall`). This figure is best seen in **colour**.

2 Sample command files for running cgcns

Command files are supported throughout the Overture. They are files that contain lists of commands. These commands can initially be saved when the user is interactively choosing options. The command files can then be used to re-run the job. Command files can be edited and changed.

In this section we present a number of command files that can be used to run cgcns. See the file `cg/cns/cmd/Readme` for a description of the command files.

2.1 Running a command file

Given a command file for cgcns such as `cylinder.cmd`, found in `cmd/cylinder.cmd`, one can type '`cgcns cylinder.cmd`' to run this command file . You can also just type '`cgcns cylinder`, leaving off the `.cmd` suffix. Typing '`cgcns noplot cylinder`' will run without interactive graphics (unless the command file turns on graphics). Note that here I assume that the `bin` directory is in your path so that the `cgcns` command is found when you type it's name. The Cgcns sample command files will automatically look for an overlapping grid in the `Overture/sampleGrids` directory, unless the grid is first found in the location specified in the command file.

When you run a command file a graphics screen will appear and after some processing the run-time dialog should appear and the initial conditions will be plotted. The program will also print out some information about the problem being solved. At this point choose `continue` or `movie mode`. Section (4.3) describes the options available in the run time dialog.

Many of the command files such as the stirring-stick problem, `stir.cmd`, can take command line arguments. For example, here are two command lines that run a problem with different grids and parameters:

```
cgcns stir -g=stir.hdf -nu=.05 -tf=1. -tp=.025
cgcns stir -g=stir2.hdf -nu=.01 -tf=1. -tp=.002 -rate=8.
```

See the comments at the top of `stir.cmd` for further explanation and examples.

2.2 Compressible flow past two offset cylinders

This example demonstrates the method CNSCAD, solution of the compressible Navier-Stokes equations using a conservative discretization with artificial diffusion.

The command file `cg/cns/twoBump.cmd` can be used with Cgcnsto compute the two-dimensional flow of a shock traveling past two offset cylinders. This example uses (a finer version) of the overlapping grid `Overture/sampleGrids/twoBump.hdf` generated using the command file `Overture/sampleGrids/twoBump.cmd`. The coefficients of viscosity and heat conduction have been set to zero so that we are solving the inviscid Euler equations.

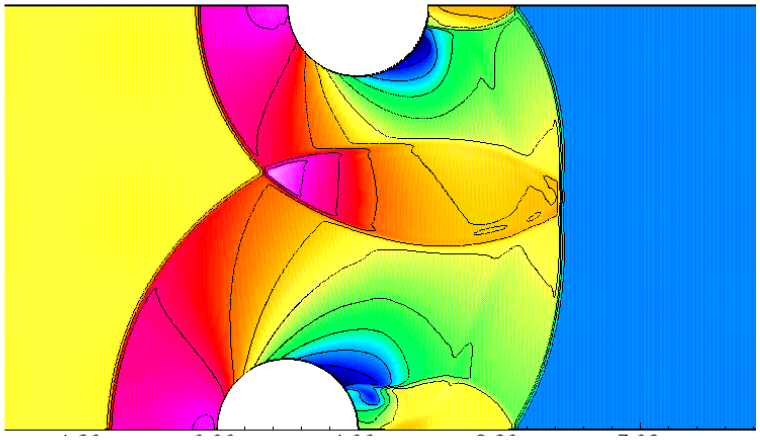

Figure 3: Solution of the compressible Euler equations: an initially plane shock, traveling from left to right, hits two offset cylinders.

2.3 Compressible flow past an airfoil

Figure 4 shows results from computing inviscid and viscous compressible flow past an airfoil in a channel. The solution is shown at time $t = 10$, starting from an impulsive start. The Mach number of the incoming flow is $M = .8$. For the viscous computation, the Reynold's number based on the chord length is around 10^5. See `OverBlown/cns/-airfoil.cmd`

Figure 5 shows results using adaptive mesh refinement.

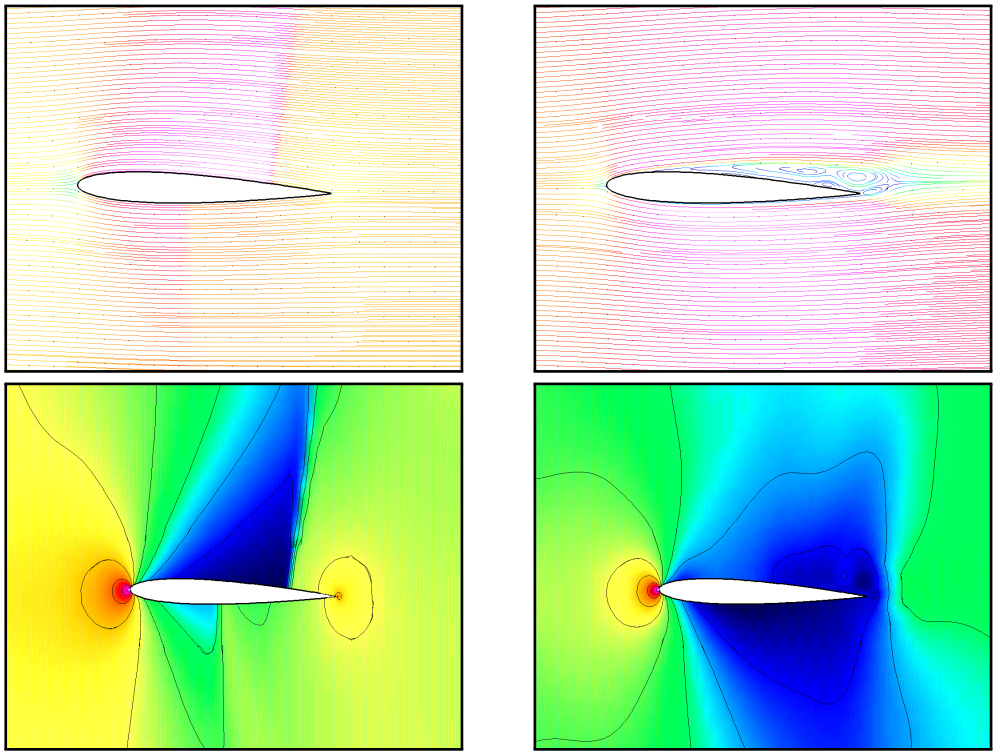

Figure 4: Compressible flow past an airfoil, showing the pressure and stream lines. The inflow Mach number is 0.8. Left: inviscid flow. Right: viscous flow.

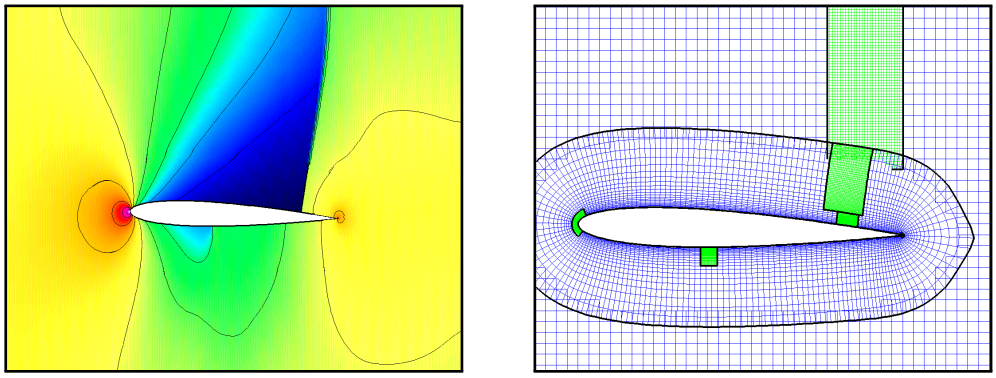

Figure 5: Compressible inviscid flow past an airfoil, showing the pressure and the grid with refinements. Mach number is 0.8.

2.4 Solving the Euler Equations with AMR

This example demonstrates the use adaptive mesh refinement with Cgcnsusing `OverBlown/cns/cicShockg.cmd`. We solve the compressible Euler equations with a conservative Godunov method (written by Don Schwendeman).

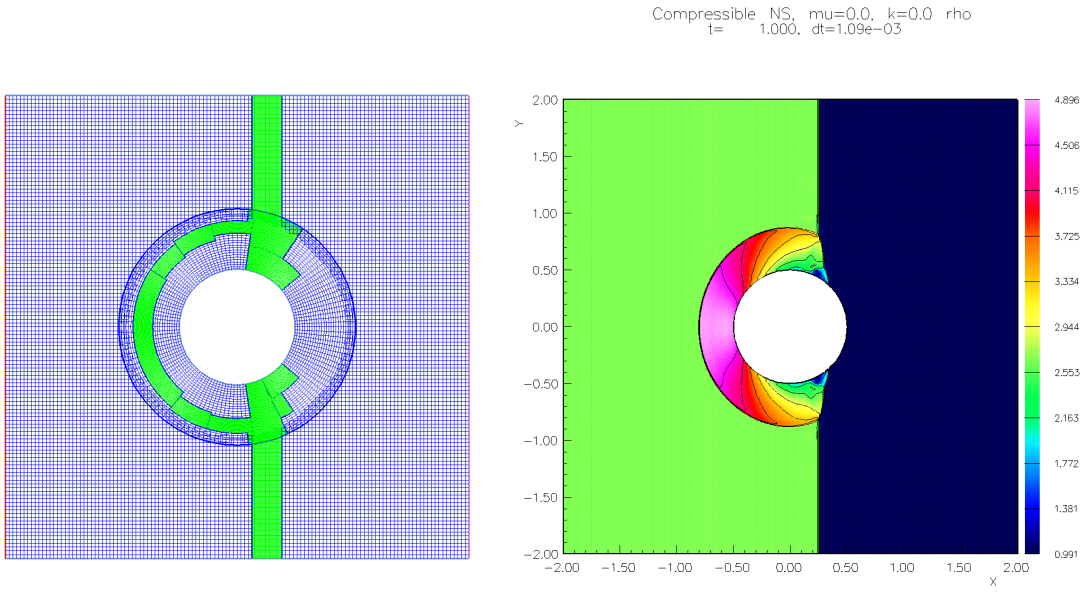

Figure 6: A shock hitting a cylinder. Adaptive mesh refinement is used to resolve the shock.

2.5 Solving the Reactive Euler Equations with AMR

In this example we solve the reactive Euler equations with AMR using `cg/cns/circleDetonation.check.cmd`. The chemistry is defined by a simple one-step reaction. An initial temperature profile is generated using an option from the user defined initial conditions, file `UserDefinedInitialConditions4.C`. A detonation forms at the hot spot, expands and reflects off the boundaries.

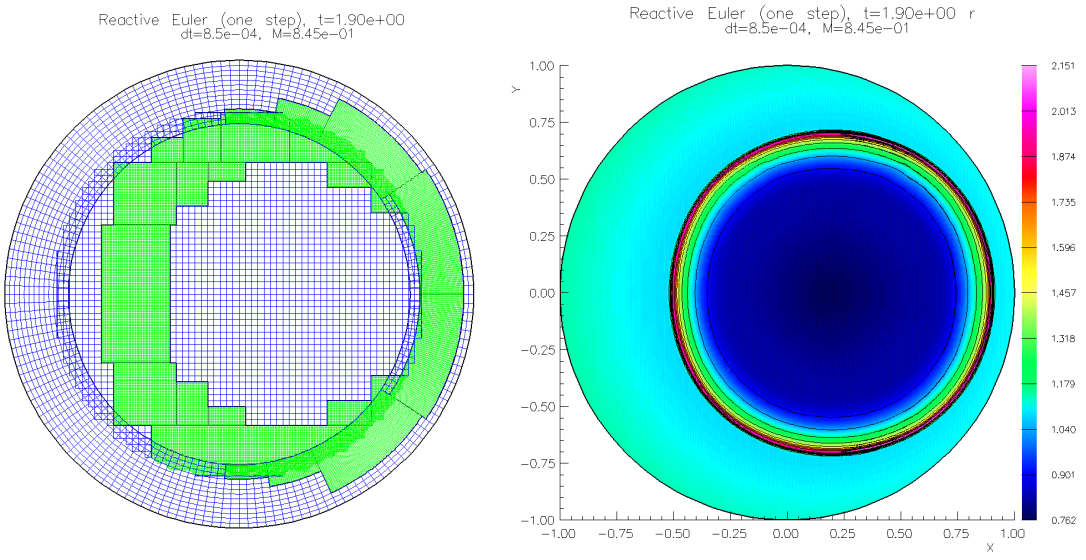

Figure 7: Solving the reactive Euler equations. Adaptive mesh refinement is used to resolve the detonation.

3 Some sample simulations

Here is a collection of interesting examples computed with the Cgcns solver.
The examples include

- Shock hitting a light rigid body, Section 3.7.

3.1 Diffraction of a cylindrically converging shock by obstacles

Finish me.

3.2 Cellular detonation

Finish me.

3.3 Detonation in an ellipse

Finish me.

3.4 Bubble collapse

Finish me.

3.5 Shock hitting multiple rigid bodies

To run this example see `cg/cns/runs/multiCylRandom/Readme`.

A shock moving from left to right impacts a collection of cylinders of random size. The fluid forces cause the cylinders to move. The cylinders are modeled as rigid bodies with a given mass density. A complex pattern of reflected and transmitted shocks results. Eventually some semblence of the incident shock appears to the right of the cylinders. This computation illustrates the use of moving grids and AMR. Collision detection has been turned on.

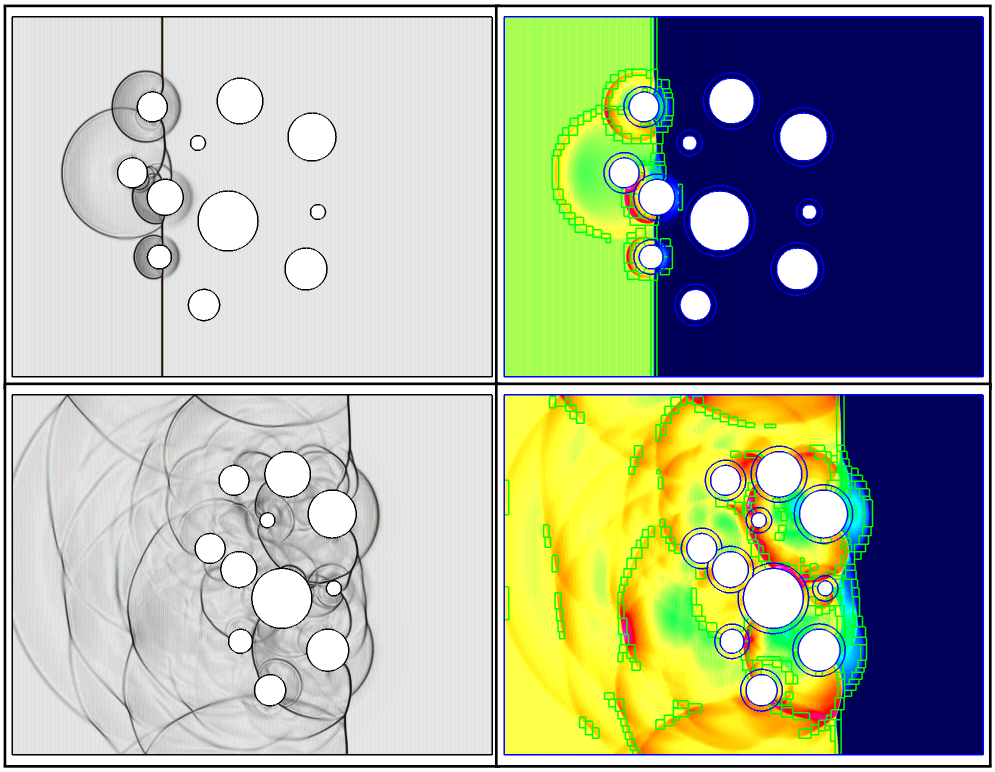

Figure 8: Shock hitting a collection of cylinders which move Schlieren images (left column) and pressure contours (right column) at times $t = 0.5$ (top) and $t = 1.5$ (bottom) on grid $\mathcal{G}_{\rm mc}^{(4\times 2)}$. The block boundaries of the refinement grids are shown superimposed on the pressure contours.

3.6 Detonation hitting multiple rigid bodies

To run this example see `cg/cns/runs/multiCylOneStep/Readme`.

A detonation moving from left to right impacts a collection of cylinders. The initial conditions for this computation are determined in the matlab script `cg/cgcns/runs/multiCylOneStep/profile.m`. This script computes the steady ZND profile for a one-dimensional detonation for various equations of state.

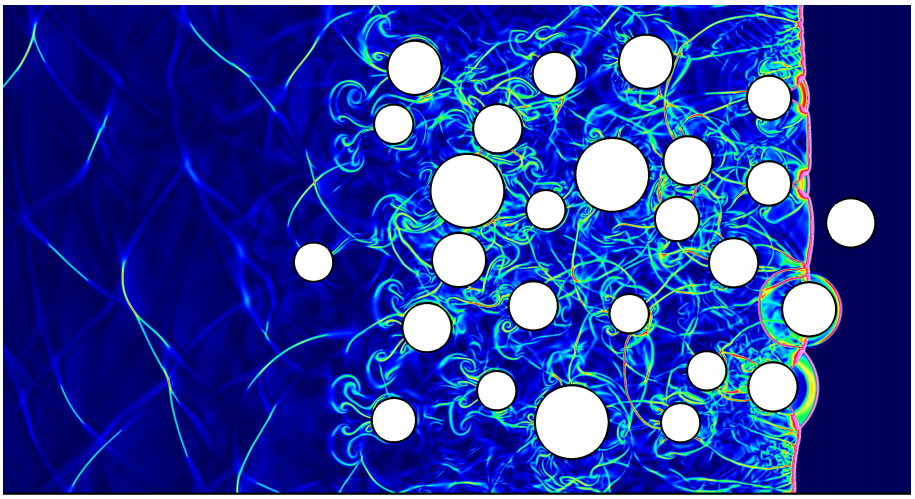

Figure 9: Detonation (one-step reaction) hitting a collection of cylinders. Colour schlieren at $t = 1.0$

3.7 Shock hitting a light rigid-body (ellipse)

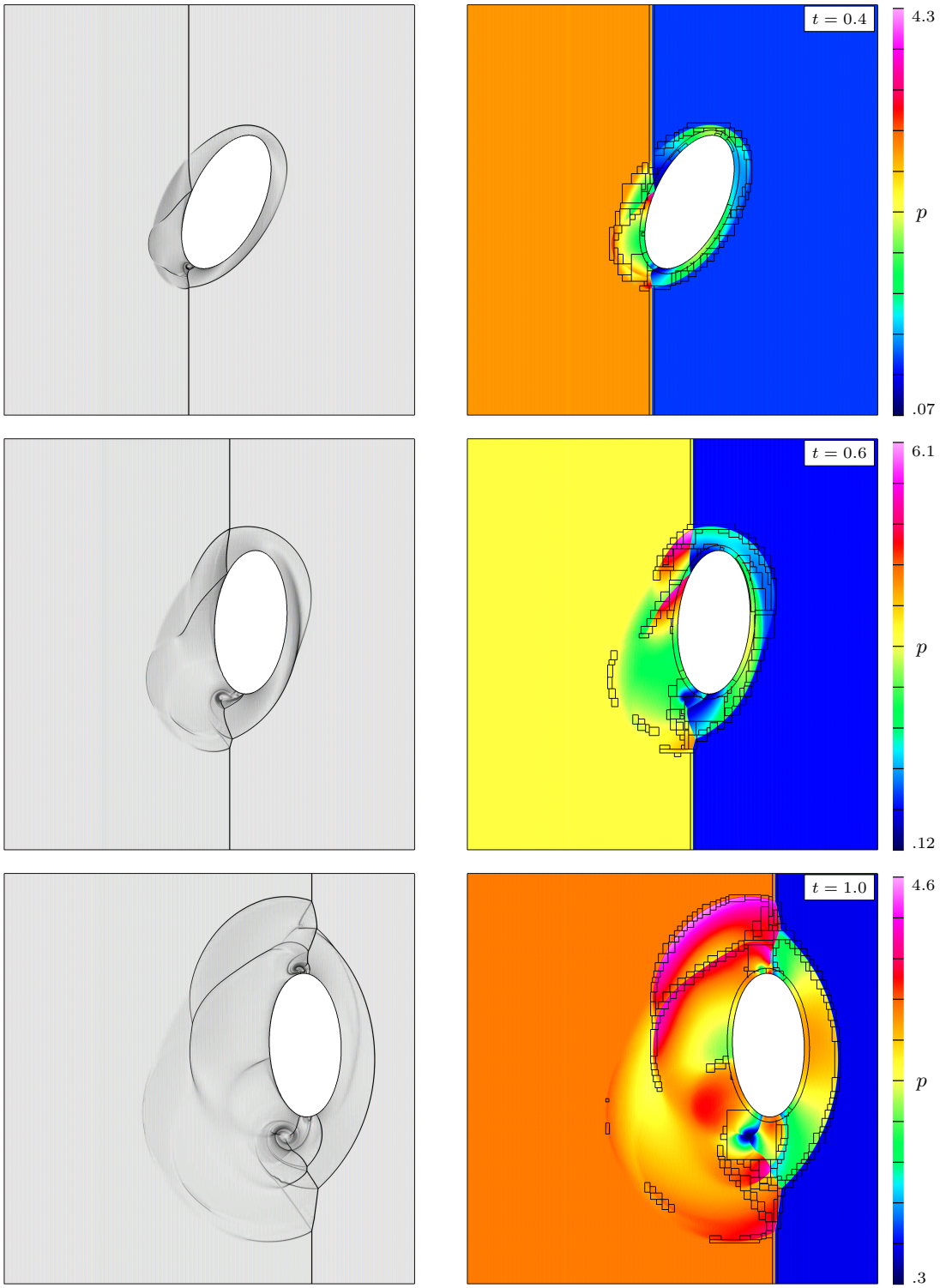

Figure 10: Shock driven zero mass ellipse. Schlieren images (left column) and pressure contours (right column) at times $t = 0.4$, $t = 0.6$ and $= 1.0$ using grid $\mathcal{G}_{\rm re}^{(16\times4)}$. The block boundaries of the refinement grids are shown superimposed on the pressure contours.

This example shows the simulation of a shock hitting a rigid body in the shape of an ellipse. This example demonstrates the new time stepping scheme that has been developed to treat the motion of "light" rigid bodies [**?**].

The simulation used the command file `cg/cns/rigidMotion.cmd` and the grid was made with the ogen script `Overture/sampleGrids/ellipseArg.cmd`.

3.8 Blast wave hitting obstacles

To run these example see `cg/cns/runs/blast/blast.cmd`.

The initial conditions consists of a circle or sphere of high pressure and density.

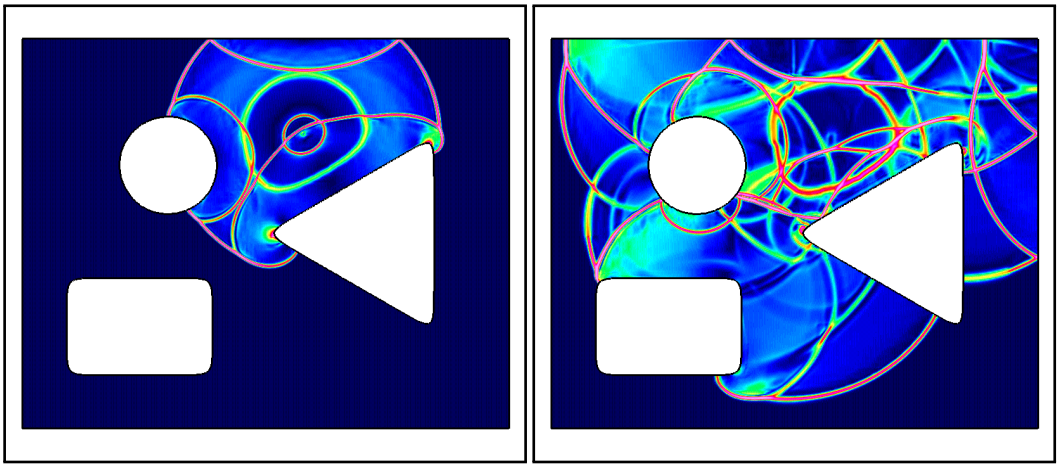

Figure 11: Blast wave hitting 3 obstacles. Reverse colour schlieren.

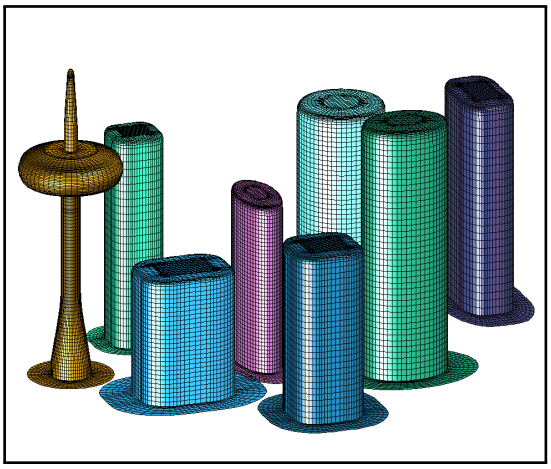

Figure 12: Grids for some buildings and tower.

Figures 16 17 18 show results from two blasts in a downtown set of buildings. Results computed with AMR. Note: (June 24, 2021) – code stopped at some point with a problem with AMR regridding – need to fix this.

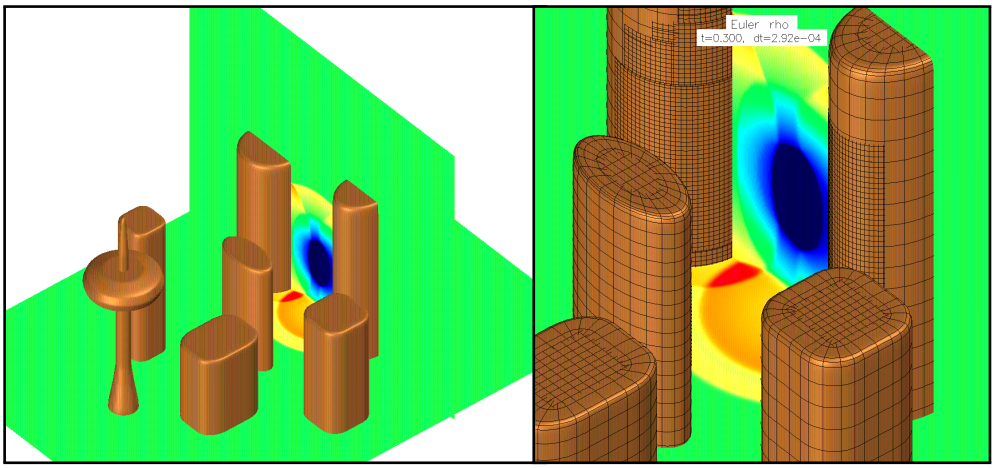

Figure 13: Blast wave hitting some buildings. Density. Note AMR grids.

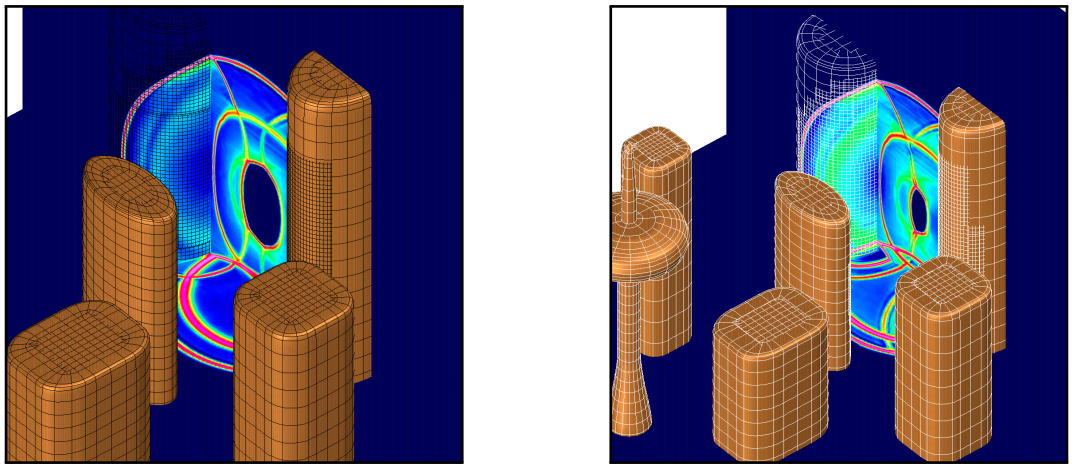

Figure 14: Blast wave hitting some buildings. Reverse colour schlieren. Note AMR grids. Grids coarsened for plotting.

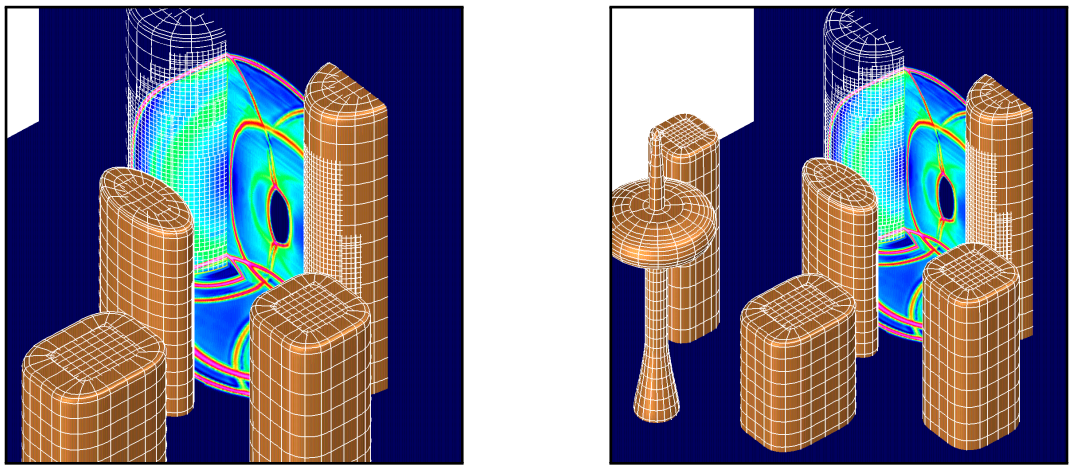

Figure 15: Blast wave hitting some buildings. Reverse colour schlieren. Note AMR grids. Grids coarsened for plotting.

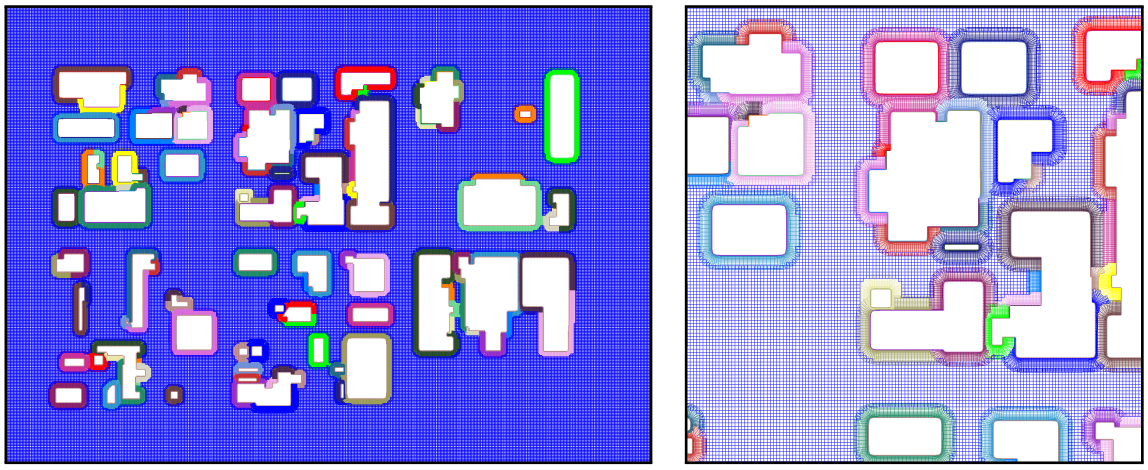

Figure 16: Grids for downtown buildings. Smoothed polygon grids are automatically constructed around polygonal building cross-sections. Note the treatment of concave corners by using multiple smoothed polygon grids on the boundary.

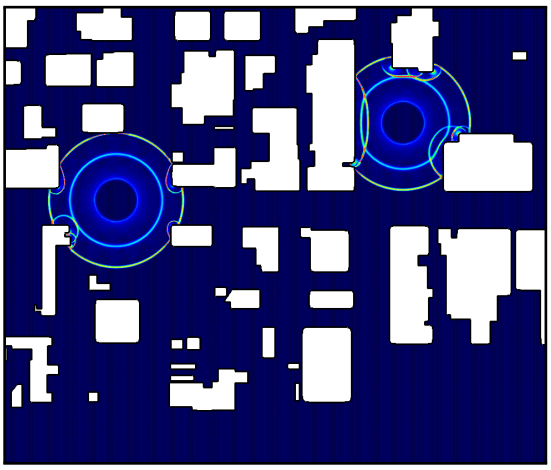

Figure 17: Blast waves hitting some downtown buildings. Reverse colour schlieren.

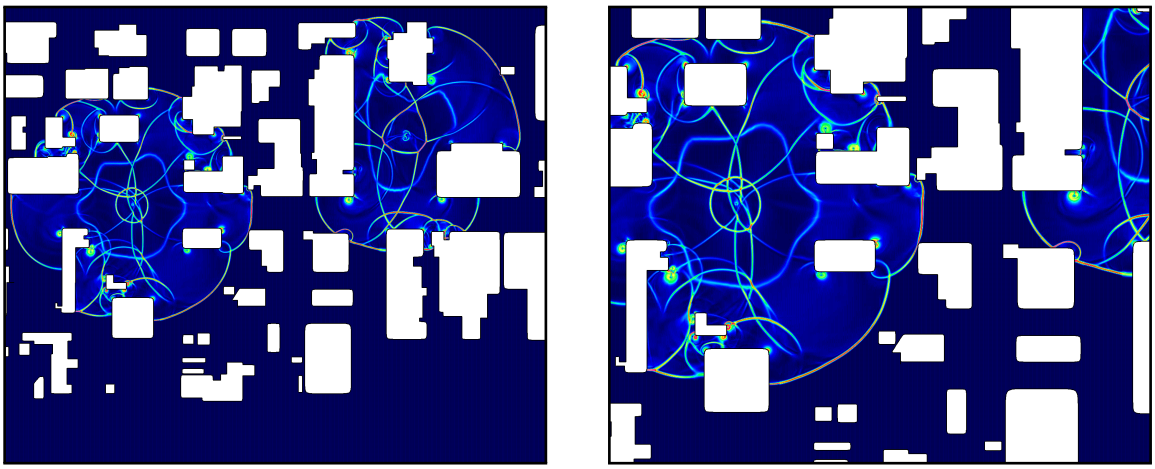

Figure 18: Blast waves hitting some downtown buildings. Reverse colour schlieren.

4 Options and Parameters

There are many options and parameters for cgcns. Be warned that not all combinations of options will work. It is best to start from an existing command file and make make minor changes.

4.1 Cgcns Setup menu

The *Cgcns Setup* dialog appears after `cgcns` is run and a grid is chosen. At this point one specifies which PDE to solve.
The options for **pde** are

compressible Navier Stokes (Jameson) : solve the compressible Navier-Stokes with a Jameson scheme.

compressible Navier Stokes (Godunov) : solve the compressible Navier-Stokes or reactive Euler equations with a Godunov scheme.

compressible Navier Stokes (multi-component) :

compressible Navier Stokes (multi-fluid) :

compressible Navier Stokes (non-conservative) : solve the compressible Navier-Stokes with a non-conservative scheme.

compressible Navier Stokes (implicit) :

steady-state compressible Navier Stokes (newton) :

The options for **reaction** are

no reactions : (default) no reactions.

one step : one-step reaction.

branching : branching reaction.

ignition and growth : ignition and growth model.

ignition and growth desensitization : ignition and growth model with desensitization.

one equation mixture fraction : .

two equation mixture fraction and extent of reaction :

one step pressure law : .

specify CHEMKIN reaction : (under development).

The options for **equation of state** are

ideal gas law : (default)

JWL equation of state :

Mie-Gruneisen equation of state :

user defined equation of state :

stiffened gas equation of state :

The options for **model** are

standard model : (default) choose this for the plain vanilla incompressible Navier-Stokes.

Boussinesq model : temperature dependent flows with buoyancy.

visco-plastic model : (under development).

two-phase flow model : (under development).

Other options include:

add advected scalars : add extra advected scalars.

add extra variables : add extra *state* variables.

axisymmetric flow with swirl : (toggle).

4.2 Cgcns Parameters Dialog and Popup menu

After choosing the PDE to solve the user will be given the opportunity to change the parameters that define the problem.

At the current time there is both a dialog menu (new) and a popup menu (old). There are some options that appear in both menus. Eventually most of the popup menu should disappear. From the main *Parameters* dialog window one can open other dialog windows such as the *Time Stepping Parameters* dialog.

The *Cgcns Parameters Dialog* push buttons are

time stepping parameters... : open the time stepping parameters dialog (see section 4.2.2).

plot options... : open the plot options dialog (see section 4.2.3).

output options... : open the output options dialog (see section 4.2.4).

pde options... : open the pde options dialog (see section 4.2.1).

boundary conditions... : open the boundary conditions dialog (see section 4.2.5).

initial conditions options... : open the initial conditions options dialog (see section 4.2.6).

forcing options... : open the forcing options dialog (see section 4.2.7).

twilight zone options... : open the twilight zone options dialog (see section 4.2.8).

showfile options... : open the showfile options dialog (see section 4.2.9).

general options... : open the general options dialog (see section 4.2.10).

adaptive grid options... : open the adaptive grid options dialog (see section 4.2.11).

moving grid options... : open the moving grid options dialog (see section 4.2.12)

plot the grid : plot the grid

display parameters : display current values of parameters

4.2.1 Compressible NS parameters Dialog (pde options...)

Here is a description of the *Compressible NS parameters Dialog* which defines options that affect the equations being solved.

The **Riemann Solver** options are

exact Riemann solver : .

Roe Riemann solver : .

HLL Riemann solver : .

aString itLabel[] = "default interpolation type", "interpolate conservative variables", "interpolate primitive variables", "interpolate primitive and pressure",""; //

The **Interpolation Type** options are

default interpolation type : .

interpolate conservative variables : .

interpolate primitive variables : .

interpolate primitive and pressure : .

The toggle buttons are

check for wall heating : .

The text commands are

mu : .

kThermal : .

thermal conductivity : .

Rg (gas constant) : .

gamma : .

Prandtl number : .

gravity : gravity vector.

nuRho : .

av2,av4 : .

aw2,aw4 : .

scoeff : strickwerdaCoeff.

slip wall boundary condition option : .

Godunov order of accuracy : .

artificial viscosity : godunov Artificial Viscosity.

heat release : .

1/(activation Energy) : .

rate constant : .

1/(activation Energy I) : .

1/(activation Energy B) : .

cross-over temperature I : .

cross-over temperature B : .

absorbed energy : .

artificial diffusion : .

boundary pressure offset : .

density lower bound : .

pressure lower bound : .

4.2.2 Cgcns Time Stepping Parameters Dialog (time stepping parameters...)

Here is a description of the *Cgcns Time Stepping Parameters* dialog. These define options related to time-stepping. The options for **method** are the available time-stepping methods. The usual choices are one of `adams PC`, `implicit` or `steady state RK-line`.

forward Euler :

adams order 2 :

adams PC : (default) Adams predictor corrector, order 2.

adams PC order 4 :

midpoint :

Runge-Kutta : (does this work?)

implicit : implicit time stepping (other options determine the exact form for this). See **implicitVariation** and **choose grids for implicit**.

variable time step PC :

steady state RK :

steady state RK-line : pseudo steady-state line solver (very memory efficient).

The *implicitVariation* options are

implicitViscous : treat viscous terms only implicitly.

implicitAdvectionAndViscous : treat viscous and advection terms only implicitly.

implicitFullLinearized : full implicit method

The *accuracy* options are

second order accurate : second-order accurate in space.

fourth order accurate : fourth-order accurate in space.

The *time accuracy* options specify the accuracy of the time stepping:

solve for steady state : time accuracy does not matter in this case.

second order accurate in time :

fourth order accurate in time :

The *predictor order* options specify the order of the predictor step for predictor corrector methods:

default order predictor :

first order predictor :

second order predictor :

third order predictor" :

fourth order predictor" :

The push buttons are

choose grids for implicit : For use with the `implicit` time stepping option. Choose which grids to integrate implicitly and which to integrate explicitly. Normally one should choose those grids with fine grid spacing (such as in boundary layers) to be implicit while a back-ground grid could be explicit.

The toggle buttons are

use local time stepping : for steady state solvers, use a different Δt for each grid point.

adjust dt for moving bodies :

use full implicit system :

apply explicit BCs to implicit grids :

The text strings are

final time : Integrate to this time.

max iterations : maximum number of iterations for steady state solvers.

cfl :Set the `cfl` parameter. The maximum time step based on stability is scaled by this factor. By default `cfl=.9`.

dtMax : Restrict the time step to be no larger than this value.

implicit factor :This value in $[0., 1.]$ is used with the implicit time-stepping. A value of .5 will correspond to a 2nd-order Crank-Nicolson approach for the viscous terms, a value of 1. will be backward-Euler and a value of 0. will be forward-Euler. See the the reference manual for more details.

recompute dt every : The time step, dt, is recomputed every time the solution is plotted/saved. In addition you may specify the maximum number of steps that will be taken before dt is recomputed. Use this if the solution is not plotted very often. See also 'slow start steps' for recomputing dt more often during a slow start.

slow start cfl : The initial time step for the slow start option is determined by this cfl value, default= 0.25.

slow start steps : Ramp the time step Δt from a small value (determined by slow start cfl) to its maximum value (as determined by the `cfl` parameter) *over this many steps* (over-rides 'slow start' based on the time interval).

slow start : Ramp the time step Δt from a small value (determined by slow start cfl) to its maximum value (as determined by the `cfl` parameter) *over this time interval*.

slow start recompute dt : during the slow start interval, recompute the time-step every this many steps.

fixup unused frequency : specifies how often to fixup unused points (which may get very large values over time that can eventually lead to the program crashing).

cflMin, cflMax :

preconditioner frequency :

number of PC corrections : specify how many correction iterations should be taken for predictor correction time-stepping (default=1).

4.2.3 Plot Options Dialog (plot options...)

Here is a description of the *Plot Options* dialog.
The *plot option* choices are

plot and wait first time :

plot with no waiting :

plot and always wait :

no plotting :

The toggle buttons are

disable plotting : disable all plotting (this can save memory when running in batch).

plot residuals : plot residual for steady state solvers.

The text commands are

times to plot : Specify the time interval between plotting (and saving in a show file). See also the **frequency to save** command.

plot iterations : For steady-state solvers, specify the number of iterations between plotting (and saving in a show file). See also the **frequency to save** command.

4.2.4 Output Options Dialog (output options...)

Here is a description of the *Output Options* dialog.
The push buttons are

output periodically to a file : output solution info to a text file.

show file options... : specify show file options (see 4.2.9).

The toggle buttons are

save a restart file : (under development)

allow user defined output : allow user defined output (see 5.5).

The text commands are

check file cutoffs : used internally for regression tests.

debug : This is a bit flag that turns on various messages. The more bits turned on, the more detailed the messages that appear. Thus a value of debug=3 (1+2) would have the first 2 bits turned on and would display few messages. A value of debug=63 (1+2+4+8+16+32) would have 6 bits turned on and would results in a lot of information.

info flag :

output format : specify the output format for printing solution values (e.g. $\%9.2e$).

4.2.5 Boundary conditions dialog (boundary conditions...)

The boundary conditions dialog can be used to construct commands that define the boundary conditions. The form of the boundary condition commands are given in section 4.4. It is often easiest to look at sample boundary conditions in existing command files and make changes to these.

As one chooses options in this dialog, a BC command is constructed in the **bc command** text command. The command is not actually applied until one chooses the **apply bc command** push button.
The *boundary* options are

all : apply the BC to all grids.

"grid name" : apply the BC to a particular grid (the grid names will appear here).

The *bc number* options allow one to apply a BC to all faces that have a particular boundary condition (assigned when the grid was made).

"bc=value" : assign the BC to all faces with this BC number (the list of BC values used on the grid are listed here).

The *set face* options allow one to apply the BC to a particular face (when assigning BCs to a given grid).

all : set all faces of the grid.

left : set the face (side,axis)=(0,0).

right : set the face (side,axis)=(1,0).

bottom : set the face (side,axis)=(0,1).

top : set the face (side,axis)=(1,1).

forward : set the face (side,axis)=(0,2).

back : set the face (side,axis)=(1,2).

The *bc* options list all the available boundary conditions. For example, these might include

noSlipWall :

inflowWithVelocityGiven :

output :

The *option* options are available qualifiers that go with a boundary condition. These include

no option :

uniform : define uniform conditions

parabolic : define a parabolic inflow, see section 4.5.1.

blasius : define a blasius profile

pressure : define a mixed BC on the pressure

jet : define a jet profile, see section 4.5.2

user defined : use a user defined boundary value, see section 5.2.

The option2 options are available second qualifiers that go with a boundary condition. These include

no option2 :

oscillate : define a time varying BC, see section 4.5.3.

ramp : define a BC that turns on slowing, see section 4.5.4.

The *extrap option* options are

polynomial extrapolation : where extrapolation is applied used normal polynomial based extrapolation.

limited extrapolation : use a limited extrapolation (for certain boundary conditions).

The push button commands are

apply bc command : apply the BC command that appears in the **bc command** text command input box.

help : print help.

plot grid : plot the grid

The text commands are

bc command : the BC command is constructed here.

default state : set the default values for each component.

order of extrap for interp neighours : set the order of extrapolation when filling in unused points next to interpolation neighbours (these points are needed in certain situations such as when a fourth-order dissipation is added to a second-order scheme).

order of extrap for 2nd ghost line : set the order of extrapolation for assigning the 2nd ghost line (these values are used in certain situations such as when a fourth-order dissipation is added to a second-order scheme).

4.2.6 Initial Conditions Options dialog (initial conditions options...)

The *plot component* option menu allows one to choose the solution component to plot when displaying the initial conditions.
The *Forcing Options* options are

no forcing :

showfile forcing : supply a forcing function from a show file.

The push buttons are

uniform flow... : open the *Uniform Flow Parameters* dialog:

The push button commands are

assign uniform state : evaluate the initial conditions.

The text commands are

uniform state : assign values to the components in the form `p=0, u=1, v=2`.

step function... : open the *Step Function Parameters* dialog.

The push button commands are

assign step function : evaluate the initial conditions.

The text commands are

state behind : assign values to the components in the form `p=0, u=2, v=1`.
state ahead : assign values to the components in the form `p=0, u=1, v=0`.
step: a*x+b*y+c*z=d : supply a, b, c, d for the equation of the plane.
step sharpness : exponent in the tanh functions used to smooth out the step. Choose a value of -1 for a true step function.

read from a show file... : open the *Read From a Show File* dialog.

The push button commands are

assign solution from show file : evaluate the initial conditions.
choose file from menu... : choose the show file from a file menu.

The toggle buttons are

use grid from show file : if true (toggle on) then use the grid in the show file. If false (toggle off) then interpolate the solution from the grid in the show file.

The text commands are

show file name : choose the name of a show file.
solution number : choose a solution number in the show file.

twilight zone... : open the *twilight zone* dialog (see section 4.2.8).

user defined... : choose a user defined initial condition (see section 5.1.

change contour plot : enter the contour plotter menu.

The text commands are

initial time : set the initial time. Normally the initial time is set to $t = 0.$ unless a solution is read from a show file (restart) in which case th initial time equals that from the show file.

4.2.7 Forcing Options Dialog (forcing options...)

Here is a description of the *Forcing Options* dialog.
The push button options are

user defined forcing : add user defined forcing (see section 5.4).

4.2.8 Twilight Zone Options Dialog (twilight zone options...)

Here is a description of the *Twilight Zone Options Dialog*.
The twilight zone *type* options are

polynomial : a polynomial function.

trigonometric : a trigonometric function.

pulse : a generalized Gaussian pulse.

The *Error Norm* options are (use this norm when reporting errors)

maximum norm :

l1 norm :

l2 norm :

The push button options are

assign polynomial coefficients : specify the coefficients in the polynomial function.

The toggle button options are

twilight zone flow : turn on the twilight zone flow.

use 2D function in 3D : use the 2D twilight zone function even in 3D.

compare 3D run to 2D : this option will adjust the equations and forcing so that a 3D run on an extruded 2D grid can be compared to the 2D computation. This includes setting the twilight-zone function to be 2D and changing the divergence damping (INS) to be two-dimensional (otherwise it is scaled in the wrong way).

assign TZ initial conditions : initial conditions are assigned the twilight zone solution.

The text commands are

degree in space : set the degree in space of the polynomial function.

degree in time : set the degree in time of the polynomial function.

frequencies (x,y,z,t) : set the frequencies in the trigonometric function.

4.2.9 Show File Options Dialog (showfile options...)

Here is a description of the *Show File Options* dialog.
The pull down menu commands are

show variables : toggle on/off variables that should be saved in the show file.

The *mode* options are

compressed : a compressed file will be smaller (especially for AMR runs that create many grids) but a compressed file may not be readable by future versions.

uncompressed :

The push button commands are

open : open a show file. You will be prompted for the name.

close : close the show file.

The text commands are

frequency to save : By default the solution is saved in the show file as often as it is plotted according to `'times to plot'`. To save the solution less often set this integer value to be greater than 1. A value of 2 for example will save solutions every 2nd time the solution is plotted.

frequency to flush : Save this many solutions in each show file so that multiple show files will be created (these are automatically handled by plotStuff). See section (4.6.1) for why you might do this.

frequency to save sequences : specify how often to save sequences (such as the residual for steady state problems).

maximum number of parallel sub-files : On a parallel machine the show file is split into parallel sub-files. If speed of the parallel writing depends on the number of processors and this value. A good value may be about 8 or 16 (depends on the number of I/O channels).

4.2.10 General Options Dialog (general options...)

Here is a description of the *General Options* dialog.
The push buttons are

pressure solver options : change the linear solver parameters for the pressure solver.

implicit time step solver options : change the linear solver parameters for implicit time stepping.

The toggle buttons are

axisymmetric flow : turn on/off axisymmetric flow.

iterative implicit interpolation : if true, solve the implicit interpolation equations by iteration rather than factoring the sparse matrix. This option should almost always be used in 3D since it can save memory and the time it takes to factor the matrix.

check for floating point errors : if true, periodically check for floating point errors such as Nan's.

The text commands are

maximum iterations for implicit interpolation : specify the maximum number of iterations for implicit interpolation (usually a value of 10 gives sufficient accuracy).

reduce interpolation width : reduce the interpolation width from the value found in the grid.

4.2.11 Adaptive Grid Options Dialog (adaptive grid options...)

Here is a description of the *Adaptive Grid Options* dialog.
The push button options are

change adaptive grid parameters : change adaptive grid parameters related to regriding (enter the Regrid menu).

change error estimator parameters : change error estimator parameters (enter the ErrorEstimator menu).

top hat parameters : set parameters for the *top hat* error function (used for testing AMR).

The toggle buttons are

use adaptive grids : turn on/off adpative grids

show amr error function : plot the amr error function with the other components.

use user defined error estimator : turn on the user defined error estimator, see section 5.3.

use default error estimator : turn on/off the default error estimator.

use front tracking : (under development).

The text commands are

error threshold : set the error threshold.

regrid frequency : specify how often to perform an AMR regrid. If you increase this value then you should also increase the number of AMR buffer zones.

truncation error coefficient : set the coefficient of the truncation term (this is defined with certain problems such as when solving the reactive Euler equations).

order of AMR interpolation : set the order of interpolation for AMR.

tracking frequency : (under development).

4.2.12 Moving Grid Options Dialog (moving grid options...)

Here is a description of the *Moving Grid Options* dialog.
The push button options are

specify grids to move : specify the grids that should move and the manner in which they move.

The toggle buttons are

use moving grids : turn on/off moving grids.

detect collisions : turn on/off collision detection.

The text commands are

minimum separation for collisions : specify the minimum separation (in grid lines) between grids during a collision. This separation is needed so that the overlapping grid remains valid.

frequency for full grid gen update : Specify how often the full grid generation algorithm is called. For moving grids, the overlapping grid generator is called at every time step. An optimized moving grid generation algorithm is used which does not minimize the overlap. Once in a while the full algorithm is used – you can change the frequency this occurs here. Choosing a value of 1 will mean the full update is always called.

4.2.13 Choosing grids for implicit time stepping.

When the option `'choose grids for implicit'` is chosen from the main parameter menu one can specify which grids should be treated implicitly or explicitly with the **implicit** time stepping option. Type a line of the form

```
<grid name>=[explicit][implicit]
```

where `<grid name>` is the name of a grid or `'all'`. Type `'help'` to see the names. Examples:

```
square=explicit
all=implicit
cylinder=implicit
```

Type `'done'` when finished.

4.3 Cgcns Run time dialog

After the equations have been specified, parameters set and initial conditions chosen, the run time dialog window will appear, see figure(1.1). Note that cgins is in the process of converting from popup menus (right mouse button) to dialog windows so sometimes you will need to look for the command in the popup menu if it is not in the dialog.

Normally one would choose **continue** to integrate to the next output time or **movie mode** to integrate until the final time.
The *plot component* option menu allows one to choose the solution component to plot.
The push button commands are

break : If running in movie mode this command will cause the program to halt at the next time to plot.

continue : compute the solution to the next time to plot.

movie mode : compute the solution to the final time without waiting. The solution will be plotted at each output time interval.

movie and save : movie mode plus save each frame as a ppm file.

contour : enter the contour plotting function in `PlotStuff`. Here you will more options to change the plot.

streamlines : enter the streamlines plotting function from `PlotStuff`.

grid : enter the grid plotting function from `PlotStuff`. If you don't first erase the contour plot then both the contours and the grid will be shown.

plot parallel dist. plot the grid showing the parallel distribution.

erase : erase the screen.

change the grid : add, remove or change existing grids. (poor man's adaptive mesh refinement).

adaptive grids... : open up a new dialog to show parameters adaptive grids.

use adaptive grids : turn adaptive grids on or off.

error threshold : specify the error threshold.

show file options... : choose show file options; e.g. open or close a show file, see section 4.2.9.

file output... : specify options for saving solutions to an ascii file (for plotting with matlab for example). There are a number of options available as to what data should be saved. See also the userDefinedOutput routine where you can customize output.

output periodically to a file : Open a file for output; specify how often to save data in the file (every step, every second step...); specify what data to save in the file (only grid 1, only values on some boundaries etc). Each time this menu item is selected a new file is opened, allowing one, for example, to save certain information every step and other information every tenth step.

close an output file : Close a file opened by the command 'output periodically to a file'.

save a restart file : save the current solution as a restart file; usually I just use the show file for restarts.

pde parameters... change PDE parameters at run time, see section 4.2.1.

general options... open the general options dialog (see section 4.2.10).

The text commands are

final time : change the value for the final time to integrate to.

times to plot : change the time interval between plotting (and output).

debug : enter an integer to turn on debugging info. This is a bit flag with debug=1 turning on just a bit of info, debug=3 (1+2) showing more, debug=7 (1+2+4) even more etc.

The **finish** button means do not compute any further, exit and save the show files etc.

4.4 Boundary Conditions

In order to compute the correct flow the user must choose the correct boundary conditions. Each physical boundary of each grid must be given a boundary condition.

The names of the available boundary conditions are

```
enum BoundaryCondition
{
  interpolation=0,
  noSlipWall,
  slipWall,
  superSonicOutflow,
  superSonicInflow,
  subSonicInflow,
  symmetry,
  dirichletBoundaryCondition,
  neumannBoundaryCondition,
  axisymmetric,
  farField
};
```

Not all boundary conditions can be used with all PDEs. Boundary conditions are specified interactively (or in a command file) by choosing the `'boundary condition'` option from the main parameters menu and then typing a string that takes one of the following forms

<grid name>(side,axis)=< boundary condition name> [,option] [,option] ...

to change the boundary condition on a given side of a given grid, or

<grid name>=<boundary condition name> [,option] [,option] ...

to change all boundaries on a given grid, or

bcNumber<num>=<boundary condition name> [,option] [,option] ...

to change all boundaries that currently have a boundary condition value equal to the integer 'num'. Here **<grid name>** is the name of the grid, side=0,1 and axis=0,1,2. **<grid name>** can also be 'all'. The optional arguments specify data for the boundary conditions:

option = 'uniform(p=1.,u=1.,...)' : to specify a uniform inflow profile

option = 'parabolic(d=2,p=1.,...)' : to specify a parabolic inflow profile

option = 'jet(r=1.,x=0.,y=0,z=0.,d=.1,p=1.,u=$U_{\max}$,v=$V_{\max}$,...)' : specify a jet inflow profile.

option = 'pressure(.1*p+1.*p.n=0.)' : pressure boundary condition at outflow

option = 'oscillate(t0=.5,omega=1.,a0=.5,a1=.5,u0=0.,v0=0.,w0=0.)' : oscillating inflow parameters

option = 'ramp(ta=0.,tb=1.,ua=0.,ub=1.,...)' : ramped inflow parameters

option = 'userDefinedBoundaryData' : use a user defined boundary value option.

option = 'mixedDerivative(1.*t+2.*t.n=3.)' : Mixed derivative on the Temperature.

Examples:

square(0,0)=inflowWithVelocityGiven , uniform(p=1.,u=1.)
square(1,0)=outflow
annulus=noSlipWall
all=slipWall
bcNumber1=noSlipWall
square(0,1)=outflow , pressure(.1*p+1.*p.n=0.)
square(0,0)=inflowWithVelocityGiven , parabolic(d=.25,p=1.,u=1.) , oscillate(t0=0.,omega=1.,a0=.5,a1=.5)
square(0,0)=inflowWithVelocityGiven , userDefinedBoundaryData
square(0,0)=inflowWithVelocityGiven , parabolic(d=.25,p=1.,u=1.) , userDefinedBoundaryData
square(0,0)=noSlipWall, uniform(u=2,v=0,T=1.), mixedDerivative(0.*t+1.*t.n=0)

The first example, `square(0,0)=inflowWithVelocityGiven`, will set the left edge of the square to be an inflow BC, while `square(1,0)=outflow` will set the right edge to be an outflow boundary. The line, `annulus=noSlipWall`, will set all physical boundaries of the annulus to be no-slip walls. Note that an annulus will normally have a branch cut and possibly an interpolation boundary. The boundary conditions on these non-physical boundaries are never changed. The command, `all=slipWall`, will make all physical boundaries slip-walls (and thus over-ride any previous changes to boundary conditions).

4.5 Data for Boundary Conditions

Some boundary conditions require 'data', such as an inflow boundary that requires values for certain quantities such as the velocity. These data values are optionally specified when the boundary condition is given. Here are some examples:

```
square(0,0)=inflowWithVelocityGiven , uniform(p=1.,u=1.)
square(0,1)=outflow , pressure(.1*p+1.*p.n=0.)
square(0,0)=inflowWithVelocityGiven , parabolic(d=.2,p=1.,u=1.), oscillate(t0=.3,omega=2.5)
```

The available options are

uniform(component=value [,component=value]...) Specify a uniform inflow profile and supply values for some of the components (components not specified will have a value of zero). Here `component0` is the name of a component such as 'p' or 'u'.

parabolic([d=boundary layer width][,component=value]...) Specify a parabolic inflow profile with a given width. See section (4.5.1) for more details.

pressure(a*p+b*p.n=c) Specify the parameters a,b,c for a pressure outflow boundary condition. Here p=pressure and p.n=normal derivative of p. **Note that a and b should have the same sign or else the condition is unstable.**

mixedDerivative(a*t+b*t.n=c) Specify the parameters a,b,c for a mixed boundary condition on the Temperature.

oscillate([t0=value][,omega=value]) Specify parameters for an oscillating inflow boundary condition. See section (4.5.3) for more details.

ramp([ta=value][,tb=value][,...]) : specify values for a *ramped* inflow. See section (4.5.4).

userDefinedBoundaryData : choose from the currently available user defined options. See section (5) for how to define your own boundary conditions.

Note that not all options can be used with all boundary conditions.

4.5.1 Parabolic velocity profile

A 'parabolic' profile can be specified as a Dirichlet type boundary condition. The parabolic profile is zero at the boundary and increases to a specified value $U_{\max}$ at a distance d from the boundary:

$$u(\mathbf{x}) = \begin{cases} U_{\max}(2 - s/d)s/d & \text{if } s \le d \\ U_{\max} & \text{if } s > d \end{cases}$$

Here s is the shortest distance between the point $\mathbf{x}$ on the inflow face to the next nearest adjacent boundary. and d is the user specified *boundary layer width*. The algorithm is quite smart at correctly determining the distance s even if the inflow boundary is covered by one or more overlapping grids (such as the pipe flow example or inlet-outlet grid).

The parabolic profile can be useful, for example, in specifying the velocity profile at an inflow boundary that is adjacent to a no-slip wall. A uniform profile would have a discontinuity at the wall.

4.5.2 Jet velocity profile

The jet option is '`jet(r=1.,x=0.,y=0,z=0.,d=.1,p=1.,u=`$U_{\max}$`,v=`$V_{\max}$`,w=`$W_{\max}$`,...)`'.

A 'jet' profile can be used to define inflow over a portion of a boundary. The jet has a a center, (x_0, y_0, z_0), a radius r, and a maximum value of $U_{\max}$ for u (or $V_{\max}$ for v or $W_{\max}$ for w) at $r = 0$:

$$u(\mathbf{x}) = \begin{cases} U_{\max} & \text{if } |\mathbf{x} - \mathbf{x}_0| \le r \\ 0 & \text{if } |\mathbf{x} - \mathbf{x}_0| > r \end{cases}$$

In 3D the jet will have a cylindrical cross section. The jet can also be defined to go to zero at it's boundary using the parameter d which defines the width of the transition layer,

$$u(\mathbf{x}) = \begin{cases} U_{\max} & \text{if } |\mathbf{x} - \mathbf{x}_0| \le r - d \\ U_{\max}[1 - (\xi/d)^2] & \text{if } r - d \le |\mathbf{x} - \mathbf{x}_0| < r \\ 0 & \text{if } |\mathbf{x} - \mathbf{x}_0| > r \end{cases}$$

Here $\xi = |\mathbf{x} - \mathbf{x}_0| - (r - d)$.

4.5.3 Oscillating values

An inflow boundary condition, `uniformInflow` or `parabolicInflow`, can be given an oscillating time dependence of the form

$$\{a_0 + a_1 \cos[2\pi\omega(t - t_0)]\} \times \{\text{uniform/parabolic profile}\} + \mathbf{u}_0$$

The parameters `omega,t0,a0,a1,u0,v0,w0` are specified with the `oscillate` option. Here $\mathbf{u}_0 = (u0, v0, w0)$.

4.5.4 Ramped Inflow

An inflow boundary condition can be ramped from one value (usually zero) to another value. The ramp function is a cubic polynomial on the interval (t_a, t_b). The polynomial is monotone increasing on this interval with slope zero at the ends. The variables (u, v, w) vary from (u_a, v_a, w_a) to (u_b, v_b, w_b). Thus the u boundary condition ramp function would be:

$$u(t) = \begin{cases} u_a & \text{for } t \le t_a \\ (t-t_a)^2(-(t-t-a)/3 + (t_b-t_a)/2)6\frac{(u_b-u_a)}{(t_b-t_a)^3} + ua & \text{for } t_a < t < t_b \\ u_b & \text{for } t \ge t_b \end{cases}$$

The ramped inflow can also be combined with the parabolic profile as in

```
square(0,0)=inflowWithVelocityGiven , parabolic(d=.25,p=1.,u=1.) , ramp(ta=0.,tb=1.,ua=0.,ub=2.)
```

to give a ramped parabolic profile.

4.6 The show file

The 'show file' is a data base file of a particular format that contains the solutions from Cgcns. The post-processing routine `plotStuff` [7] knows how to read this file and find all the solutions and the different grids if the grids are moving or adaptive. The show can be looked at by typing **'plotStuff file.show'** or just **'plotStuff file'**, where `file.show` is the name that you gave to the show file when running cgins. The program `plotStuff` is found in `Overture/bin`.

4.6.1 Flushing the show file

It is not possible to look at results in a show file while the program is running and the show file is open and being written to. As a result, if the program crashes for some reason you will not be able to look at the results. To overcome this problem it is possible to automatically save multiple show files, with each show file containing one or more solutions. The number of solutions saved in each show file is determined by the frequency the show file is flushed. Use the **'frequency to flush'** option to specify how many solutions should be saved in each show file. The files are named 'file.show', 'file.show1', 'file.show2' etc. where 'file.show' was the name given to the show file. The plotStuff program will automatically read all these different files if you just type 'plotStuff file.show'.

It is thus possible to look at the solutions when cgins crashes or while it is still running. Only the most recent solutions that belong to the most recent (open) show file will be unavailable.

4.7 Restarts

The easiest way to restart is to choose your initial conditions to come from the show file that you saved in a previous run, see section 4.2.6. The program will let one choose any solution in the show file as an initial condition. Remember to rename the show file from the previous run so that it doesn't get over-written before you have a chance to read from it.

You can also restart using an explicit **restart file**. To do this you need to turn on the saving of a restart file, see section (4). In this case cgins will write a restart file every time the solution is output. Actually, to be safe, two files are created named 'ob1.restart' and 'ob2.restart'. This is in case the program crashes while the restart file is being written. Usually both files will be valid as use for restart files.

To **read the restart file** you simply specify this option and the file to use when assigning initial conditions, see section (4.2.6).

5 User defined functions

Here is a list of functions that can be changed by a user. After rewriting any of these files, compile and link cgcns with the new file.

5.1 User defined initial conditions

The function **userDefinedInitialConditions** in `cg/common/src/userDefinedInitialConditions.C` defines initial conditions. Rewrite this function to define arbitrary initial conditions. This function is accessed when interactively setting parameters in the **'initial conditions'** menu under **'user defined'**.

5.2 User defined boundary values

The functions **chooseUserDefinedBoundaryValues, userDefinedBoundaryValues** in `cg/common/src/userDefinedBoundaryValues.C` define values for boundary conditions. Change these functions in order to define new right-hand-side values for boundary conditions. For example, you may want to define the inflow velocity profile to have a certain shape and/or time dependence. The `chooseUserDefinedBoundaryValues` is accessed when you specify boundary conditions and choose the `userDefinedBoundaryData` option.

5.3 User defined error estimator

Change the file `cg/common/src/userDefinedErrorEstimator.C` to provide your own AMR error estimator. You will need to ``use user defined error estimator'', see section 4.2.11.

5.4 User defined forcing

Change the file `cg/common/src/userDefinedForcing.C` to add forcing functions to the equations.

5.5 User defined output

Change the file `cg/common/src/userDefinedOutput.C` to output information at each time step. You will need to ``allow user defined output'', see section **??**.

5.6 User defined grid motion

Change the file `cg/common/movingsrc/userDefinedMotion.C` to define a grid motion.

Change the file `cg/common/movingsrc/userDefinedDeformingSurface.C` to define a deforming surface motion.

6 Hints for running cgcns

FINISH ME

- Start out with a simple problem on a coarse grid so that the problem can be quickly run to determine if you have the boundary conditions correct etc.
- Start out by taking only a few time steps and looking at the solution to see if it looks correct.
- The rule of thumb for choosing the viscosity ν is that if the velocities are order 1 and the domain is order 1 then $\nu > h_{\max}^2$, where $h_{\max}$ is the maximum grid spacing as reported by cgcns when it runs. This comes from the minimum scale result as discussed in section (**??**).
- If you want to use as small a viscosity as possible then set $\nu = 0$ and use artificial viscosity as discussed in sections (4.2.1,**??**).
- If cgcns blows up it could be the time step is not computed correctly. Reduce the cfl parameter (default is .9) to a value like .5 or .25 to see if this is the problem. There are known problems with the time step determination for implicit time stepping and a large viscosity (relative to the grid spacing).

7 Trouble Shooting and Frequently asked Questions

Question: I wonder what is the meaning of this error and what can I do to avoid it.

```
computeNumberOfStepsAndAdjustTheTimeStep:ERROR: time step too small? dtNew=1.560026e-61, timeInterval=7.314
 t=3.699269e+00, tFinal=6.000000e+00, nextTimeToPrint=3.700000e+00, tPrint=1.000000e-03
error
Overture::abort: I am now going to purposely abort so that you can get a traceback from a debugger
Abort
```

FINISH ME

8 Post-processing: Reading a show file and computing some Aerodynamic Quantities

The program `bin/aero.C` shows how to read a show file that has been generated by cgcns and access the solution values stored there. This program can then be used to plot the pressure coefficient on the surface of a body and to compute the lift and drag on a body.

The file `aero.C` can be altered to compute other quantities that may be of particular interest to your application. All information about the grid, solutions and cgins parameters are accessible from the show file. You could, for example use this program to output the solution values to a data file format suitable for some other plotting or analysis program.

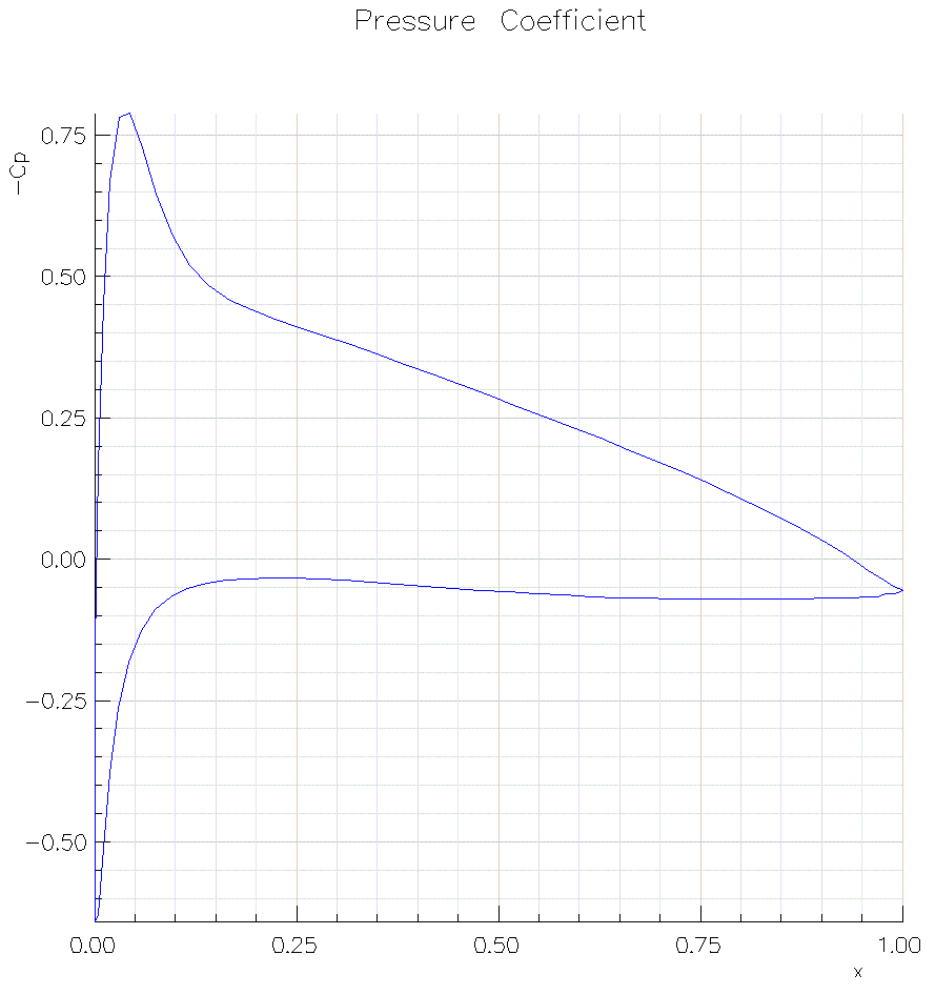

Figure 19: The areo.C program can be used to read a show file generated by cgins and compute the coefficient of pressure on the surface of a body.

References

[1] D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented software system for solving partial differential equations in serial and parallel environments*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, 1997, p. .

[2] D. L. Brown, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented framework for solving partial differential equations*, in Scientific Computing in Object-Oriented Parallel Environments, Springer Lecture Notes in Computer Science, 1343, 1997, pp. 177–194.

[3] W. D. Henshaw, *Overture: An object-oriented system for solving PDEs in moving geometries on overlapping grids*, in First AFOSR Conference on Dynamic Motion CFD, June 1996, L. Sakell and D. D. Knight, eds., 1996, pp. 281–290.

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

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

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

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

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

[9] W. D. Henshaw and D. W. Schwendeman, *An adaptive numerical scheme for high-speed reactive flow on overlapping grids*, J. Comput. Phys., 191 (2003), pp. 420–447.

[10] ——, *Moving overlapping grids with adaptive mesh refinement for high-speed reactive and non-reactive flow*, J. Comput. Phys., 216 (2006), pp. 744–779.

[11] ——, *Parallel computation of three-dimensional flows using overlapping grids with adaptive mesh refinement*, Research Report UCRL-JRNL-236681, Lawrence Livermore National Laboratory, 2008. Accepted, Journal of Computational Physics.

Index