

# Eilmer4: A Computational Tool for High-Enthalpy Flow Simulation

Peter Jacobs, Rowan Gollan (as co-chief gardeners)  
Kyle Damm, Elise Fahy, Daniel Potter  
James Burgess, Heather Muir  
and many others, as listed on the final slide...  
School of Mechanical Engineering, UQ

08 Sep 2016

Motivation and History

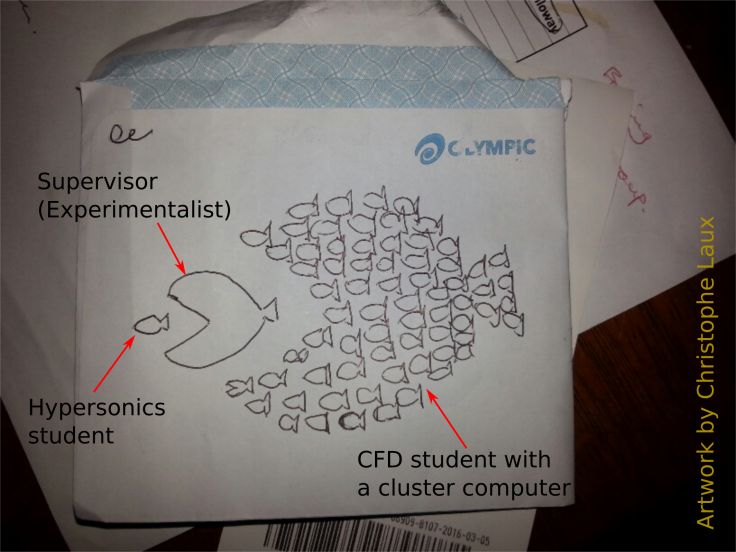
Gas dynamic formulation and code implementation

Example – hemisphere in shock tube

List of Contributors

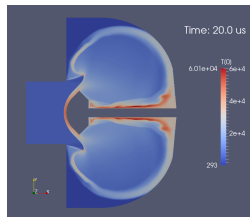


# Motivation – Computational Fluid Dynamic Tools



Artwork by Christophe Laux

# Eilmer in a nutshell



- ▶ Eulerian/Lagrangian description of the flow (finite-volume, 2D axisymmetric or 3D).
- ▶ Transient, time-accurate, optionally implicit updates for steady flow.
- ▶ Shock capturing plus shock fitting boundary.
- ▶ Multiple block, structured and unstructured grids.
- ▶ Parallel computation on a cluster computer, using MPI in Eilmer2,3 and shared memory in dgd/Eilmer.
- ▶ High-temperature nonequilibrium thermochemistry (GPU).
- ▶ Dense-gas thermodynamic models and rotating frames of reference for turbomachine modelling.
- ▶ Turbulence models: Baldwin-Lomax and  $k-\omega$ .
- ▶ Coupling to radiation and ablation codes for aeroshell flows.
- ▶ ...plus conjugate heat transfer and MHD

# Origins

- ▶ in the late 1980s, the state of the art for scramjet simulations involving reactive flow was JP Drummond *SPARK* code
- ▶ Flow solver component based on Bob McCormack's (1969) finite-difference shock-capturing technique.
- ▶ All configuration hard-coded into the Fortran source code and compiled to run on a Cray supercomputer.
- ▶ In the 1980s, a new CFD technology (upwind flux) was being developed by the applied mathematics people and parallel computing environments were being developed by the computer science people (cluster computers).
- ▶ Dec 1990: following a CFD lesson on the chalk-board from Bob Walters and Bernard Grossman, *cns4u* was started with the intention to be like SPARK but with new technology

# Development of Eilmer

- ▶ 1993 built *sm3d*, a space-marching code for 3D scramjet flows
- ▶ 1995 through 1999: the postgrad years expanded scope of experimentation and application
- ▶ 1996: code reformulation around fluxes (frequent discussions with Mike Macrossan); all code still in C with a preprocessor having a little command interpreter built in.
- ▶ 1997: discovered scripting languages Tcl and Python
- ▶ May 2003: *scriptit.tcl* provided fully programmable environment for simulation-preparation.
- ▶ Aug 2004: *Elmer* began as a hybrid code using Python and C.
- ▶ Jun 2005: rewrite of *Elmer(2)* in C alone.
- ▶ Jul 2006: rewrite *Elmer2* in C++ and, in 2008, call it *Eilmer3*. Class-based implementation was easier to extend.



- ▶ Jun 2015+: rebuild in the D and Lua programming languages.
- ▶ Heather Muir has been working on the unstructured-grid generator. based on the paving algorithm.

# Mathematical gas dynamics (in differential form)

Conservation of mass:

$$\frac{\partial}{\partial t} \rho + \nabla \cdot \rho \mathbf{u} = 0 \quad (1)$$

Conservation of species mass:

$$\frac{\partial}{\partial t} \rho_i + \nabla \cdot \rho_i \mathbf{u} = -(\nabla \cdot \mathbf{J}_i) + \dot{\omega}_i \quad (2)$$

Conservation of momentum:

$$\frac{\partial}{\partial t} \rho \mathbf{u} + \nabla \cdot \rho \mathbf{u} \mathbf{u} = -\nabla p - \nabla \cdot \left\{ -\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^\dagger) + \frac{2}{3} \mu(\nabla \cdot \mathbf{u}) \boldsymbol{\delta} \right\} \quad (3)$$

Conservation of total energy:

$$\begin{aligned} \frac{\partial}{\partial t} \rho E + \nabla \cdot \left( e + \frac{p}{\rho} \right) \mathbf{u} = & \nabla \cdot \left[ k \nabla T + \sum_{s=1}^{N_v} k_{v,s} \nabla T_{v,s} \right] + \nabla \cdot \left[ \sum_{i=1}^{N_s} h_i \mathbf{J}_i \right] \\ & - \left( \nabla \cdot \left[ \left\{ -\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^\dagger) + \frac{2}{3} \mu(\nabla \cdot \mathbf{u}) \boldsymbol{\delta} \right\} \cdot \mathbf{u} \right] \right) - Q_{\text{rad}} \quad (4) \end{aligned}$$

Conservation of vibrational energy:

$$\frac{\partial}{\partial t} \rho_i e_{v,i} + \nabla \cdot \rho_i e_{v,i} \mathbf{u} = \nabla \cdot [k_{v,i} \nabla T_{v,i}] - \nabla \cdot e_{v,i} \mathbf{J}_i + Q_{T-v_i} + Q_{V-v_i} + Q_{\text{Chem}-v_i} - Q_{\text{rad},i} \quad (5)$$



## More maths...

*Thermodynamic model of the gas...*

*Finite-rate chemical kinetics...*

*Radiation energy exchange...*

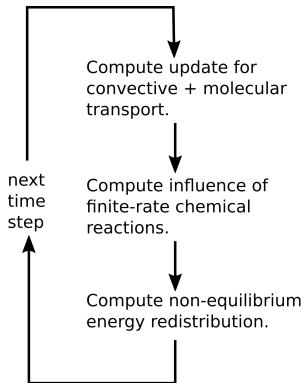
*Boundary conditions...*

Features:

- ▶ 3D from the beginning, 2D as a special case
- ▶ structured- and unstructured-meshes for complex geometries
- ▶ refined thermochemistry
- ▶ moving meshes (Jason Qin and Kyle Damm)
- ▶ simplified and generalized boundary conditions
- ▶ coupled heat transfer
- ▶ shared-memory parallelism for multicore workstation use
- ▶ block-marching for speed (nozfr and nozzle design)

# Code structure

- ▶ D language data storage and solver, with embedded Lua interpreters for preprocessing, user-controlled run-time configuration in boundary conditions and source terms and thermochemical configuration.



```
for s=1 to n do:  
  clear flux data  
  apply pre-reconstruction action  
  detect shock points  
  reconstruct flow data at cell interfaces  
  compute convective fluxes  
  apply post-convective-flux action  
  apply pre-spatial-derivative action  
  compute spatial derivatives  
  apply post-diffusion flux action  
  add source terms, if any  
  compute time derivatives of conserved quantities  
  update cell-average conserved quantities for stage s  
  decode conserved quantities to all flow quantities
```

# Verification and Validation Examples

## *Verification:*

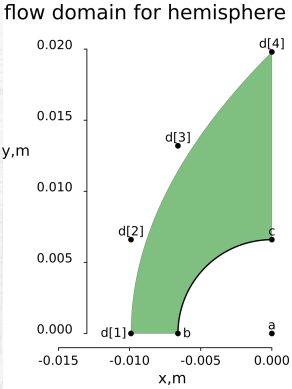
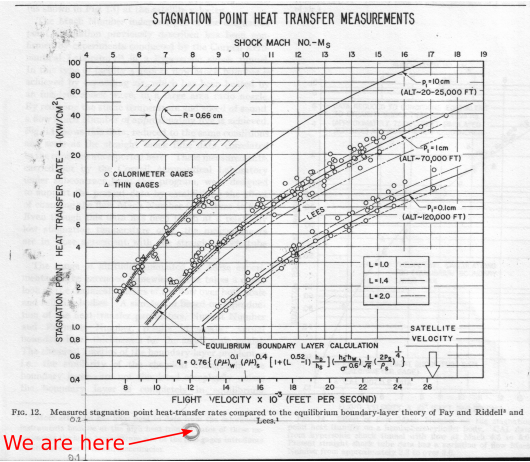
- ▶ Are we solving the equations correctly?
- ▶ Compare with numerical solutions from other codes.
- ▶ Manufactured solution that we must match (using special source terms and boundary conditions).

## *Validation:*

- ▶ Are we solving the correct gas-dynamic equations?
- ▶ Compare with experimental measurements.

# Example: hemispherical probe in shock-tube flow

- ▶ Rose & Stark, 1958
- ▶ Approximate as a 2D, axisymmetric flow around a hemisphere.



## Input script – gas model and flow

```
config.title = "Rose and Stark experiment"  
R = 6.6e-3 -- nose radius, metres  
-- free stream conditions taken from Table 1, Entry 5  
nsp, nmodes, gmodel = setGasModel('cea-lut-air.lua')  
p_init = 6.7 -- Pa  
p_inf = 535.6 -- Pa  
T_inf = 2573.5 -- K  
vx_inf = 2436.5 -- m/s  
inflow = FlowState:new{p=p_inf, T=T_inf, velx=vx_inf}  
initial = FlowState:new{p=p_init, T=T_inf}
```

### Notes:

- ▶ user's input script is Lua source code
- ▶ arguments to function calls delimited by ()
- ▶ tables delimited by {}
- ▶ object model by convention as described in Ierusalimsky's book "Programming in Lua"

# Input script – geometry definition

```
-- Set up the geometry for defining the grid
a = Vector3:new{x=0.0, y=0.0}
b = Vector3:new{x=-R, y=0.0}
c = Vector3:new{x=0.0, y=R}
d = { Vector3:new{x=-1.5*R, y=0.0}, Vector3:new{x=-1.5*R, y=R},
      Vector3:new{x=-R, y=2*R}, Vector3:new{x=0.0, y=3*R} }
-- Set up surface and grid
sphere_edge = Arc:new{p0=b, p1=c, centre=a}
psurf = makePatch{north=Line:new{p0=d[#d], p1=c}, south=Line:new{p0=d[1], p1=b},
                 east=sphere_edge, west=Bezier:new{points=d}}
cf_radial = RobertsFunction:new{end0=false, end1=true, beta=1.2}
grid = StructuredGrid:new{psurface=psurf, niv=ni+1, njv=nj+1,
                          cfList={north=cf_radial, south=cf_radial}}
```

## Notes:

- ▶ Fully-parametric grid generator is available.
- ▶ Table entries are mostly named. This is an advantage for large numbers of parameters and helps to make your input script self-documenting.
- ▶ Also, could import grids. Good for complex geometries because you may use your favourite gridding tool.

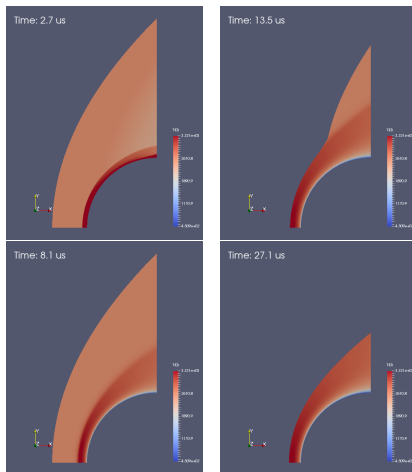
# Input script – flow-domain with boundary conditions

```
blk = SBlockArray{grid=grid, fillCondition=initial, label='blk',  
                 bcList={west=InFlowBC_ShockFitting:new{flowCondition=inflow},  
                         east=WallBC_NoSlip_FixedT:new{Twall=296.0},  
                         north=OutFlowBC_Simple:new{}},  
                 nib=1, njb=4}  
  
dofile("sketch-domain.lua")
```

## Notes:

- ▶ May define many blocks on a single grid.
- ▶ We attach boundary conditions to the domain and specify the initial flow condition.
- ▶ Boundary conditions default to class `WallBC_WithSlip`.
- ▶ Some boundary conditions need extra information.

# Results – evolution of the temperature field

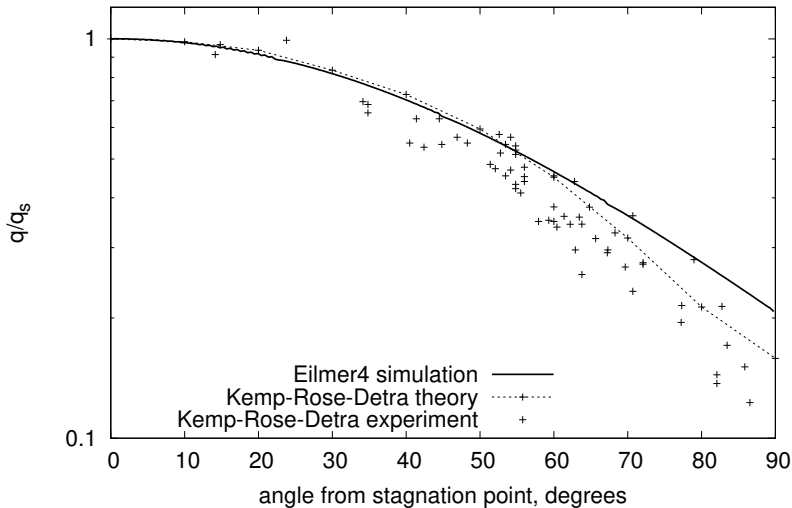


- ▶ Characteristic time
$$\tau = \frac{R}{V_{x,\infty}} = 2.7 \mu\text{s}.$$
- ▶ Start viscous effects at  $2\tau$ .
- ▶ Start shock-fitting at  $3\tau$ .
- ▶ Run simulation to  $10\tau$ .
- ▶ Temperature range shown is 451 K to 3331 K.

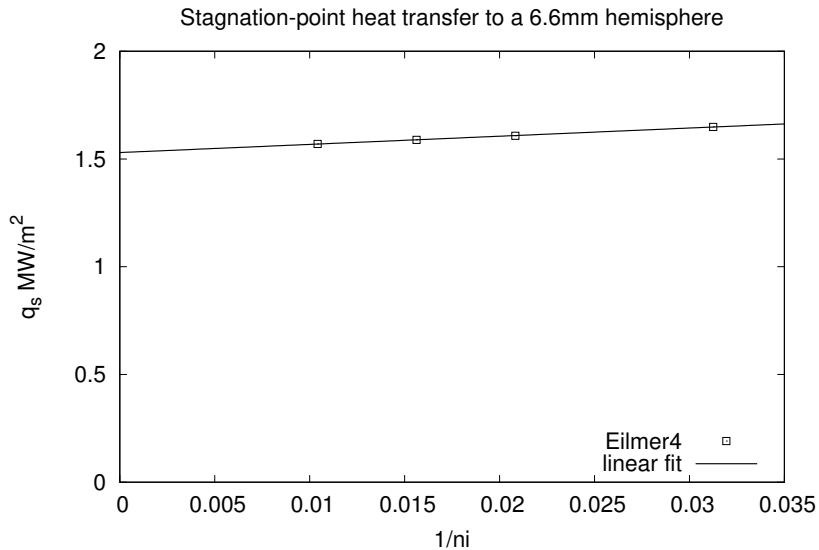


# Results – heat transfer distribution around sphere

Normalised heat transfer to R6.6mm sphere with  $Ms=8$



# Results – stagnation-point heat transfer



# The Many Contributors

Ghassan Al'Doori, Nikhil Banerji, Justin Beri, Peter Blyton, Daryl Bond, Arianna Bosco, Djamel Boutamine, Laurie Brown, James Burgess, David Buttsworth, Wilson Chan, Sam Chiu, Chris Craddock, Brian Cook, Jason Czaplá, Kyle Damm, Andrew Dann, Andrew Denman, Zac Denman, Luke Doherty, Elise Fahy, Antonia Flocco, Delphine Francois, James Fuata, Nick Gibbons, David Gildfind, Richard Goozéé, Sangdi Gu, Stefan Hess, Jonathan Ho, Carolyn Jacobs, Ingo Jahn, Chris James, Ian Johnston, Ojas Joshi, Xin Kang, Rainer Kirchhartz, Sam Lamboo, Steven Lewis, Tom Marty, Matt McGilvray, David Mee, Carlos de Miranda-Ventura, Luke Montgomery, Heather Muir, Jan-Pieter Nap, Brendan O'Flaherty, Andrew Pastrello, Paul Petrie-Repar, Jorge Sancho Ponce, Daniel Potter, Jason (Kan) Qin, Deepak Ramanath, Andrew Rowlands, Michael Scott, Umar Sheikh, Sam Stennett, Ben Stewart, Joseph Tang, Katsu Tanimizu, Augustin Tibere-Inglesse, Pierpaolo Toniato, Paul van der Laan, Tjarke van Jindelt, Anand Veeraragavan, Jaidev Vesudevan, Han Wei, Mike Wendt, Brad (The Beast) Wheatley, Vince Wheatley, Lachlan Whyborn, Adriaan Window, Hannes Wojciak, Fabian Zander, Mengmeng Zhao