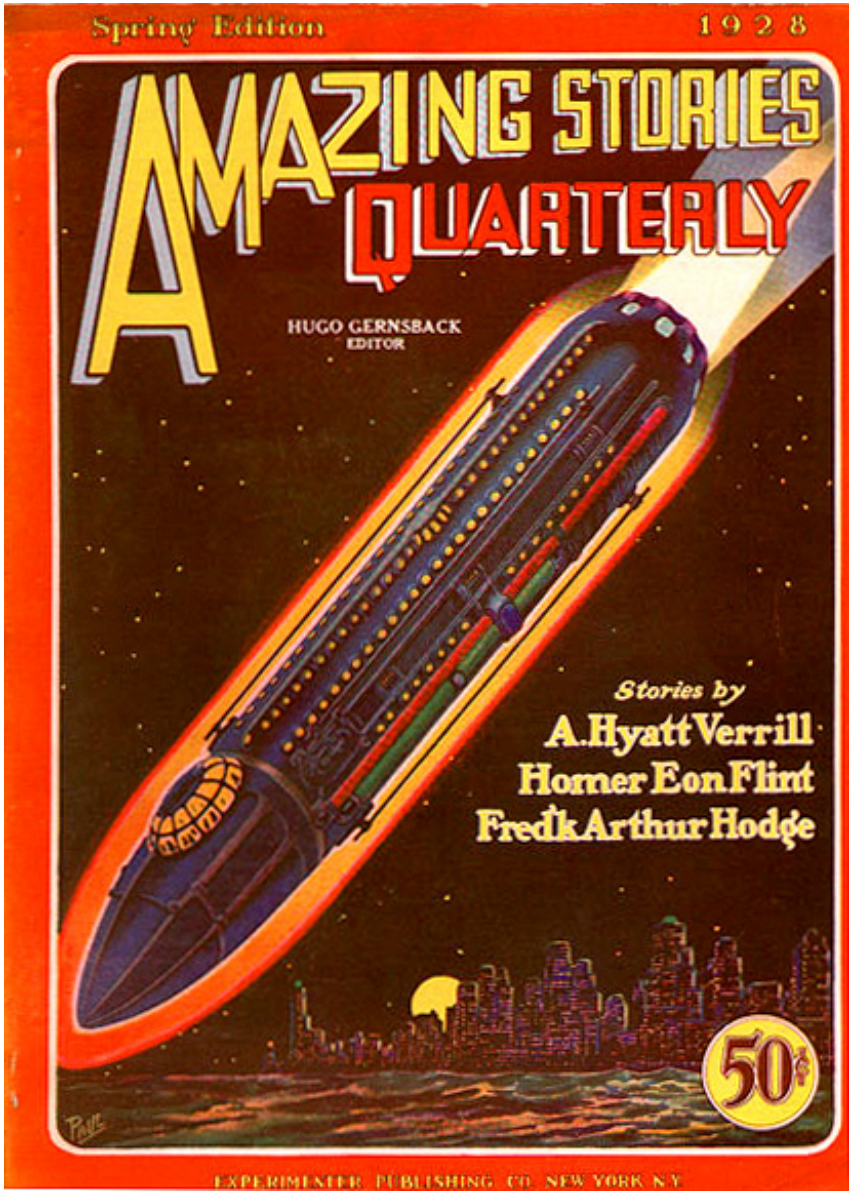
Eilmer4: the next step in the UQ simulation codes for high-enthalpy flows

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, University of Queensland

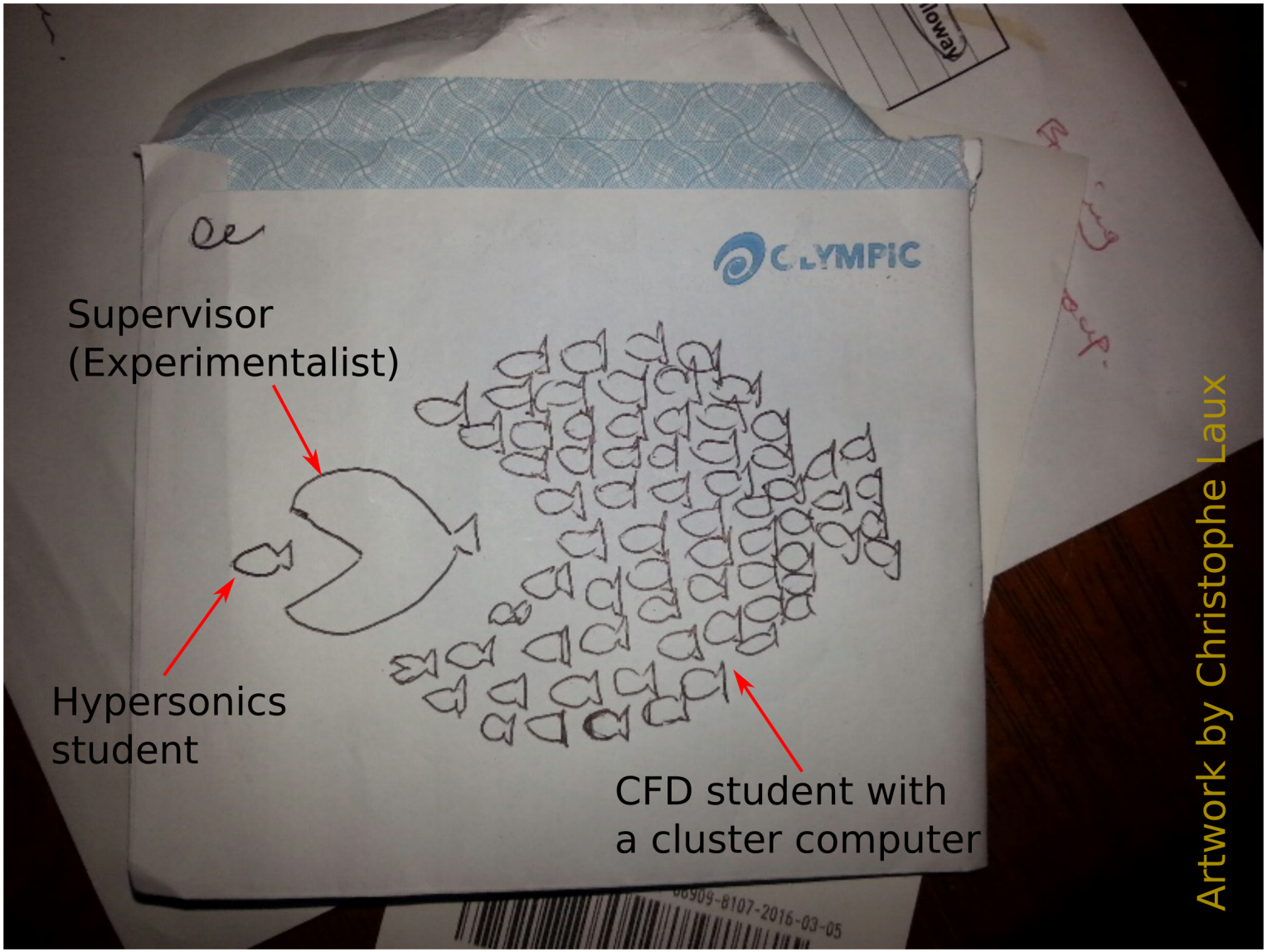
06 December 2016

Motivation – High-enthalpy Flows

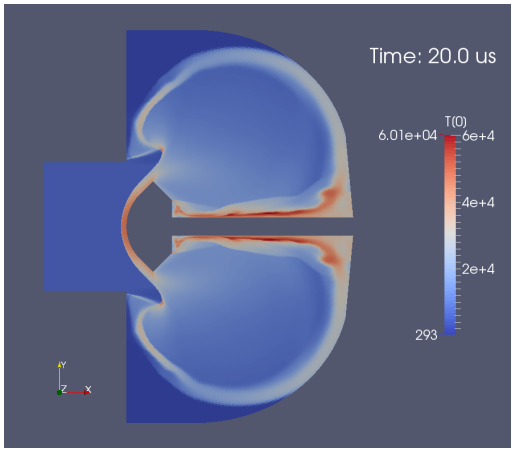


- An application of high-enthalpy flow from your grand-parents' days.
- Not much has changed.

Motivation – Computational Fluid Dynamic Tools



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/Eilmer4.
- High-temperature nonequilibrium thermochemistry (GPU).
- Dense-gas thermodynamic models and rotating frames of reference for turbomachine modelling.
- Turbulence models: Baldwin-Lomax and k-ω.
- 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.

Eilmer – Let's do it right, again.

Fred Brooks, in the "Mythical Man-Month: Essays on software engineering"

> *Sooner or later the first system is finished, and the architect, with firm confidence and a demonstrated mastery of the class of systems, is ready to build a second system. ...*
> *This second is the most dangerous system a man ever designs. ...*
> *The general tendency is to over-design the second system, using all the ideas and frills that were cautiously sidetracked on the first one.*

We're OK, this is not our *second* system.
cns4u, mbcns, mbcns2, Elmer, Elmer2, Eilmer3 ... Eilmer4.

Eilmer4 – think big, but control the complexity.

- 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:

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

Conservation of species mass:

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

Conservation of momentum:

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

Conservation of total energy:

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

Conservation of vibrational energy:

$$\frac{\partial}{\partial t}\rho_i e_{v,i} + \nabla \cdot \rho_i e_{v,i}\mathbf{u} = \nabla \cdot [k_{v,i}\nabla T_{v,i}] - \nabla \cdot e_{v,i}\mathbf{J}_i + Q_{T-V_i} + Q_{V-V_i} + Q_{\mathrm{Chem}-V_i} - Q_{\mathrm{rad}_i} \tag{5}$$

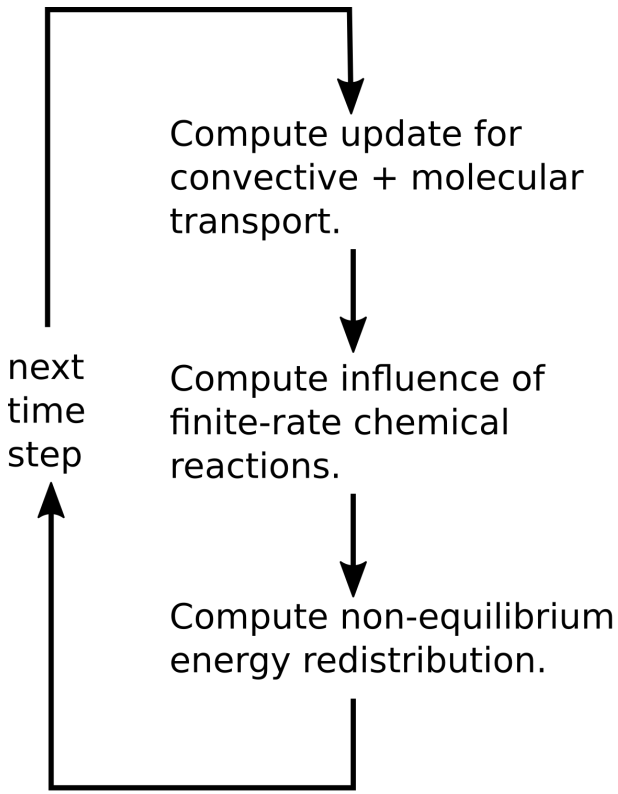
More maths...

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 (nenzfr 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 derivatves
- 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

Collecting the low-hanging fruit of parallelism

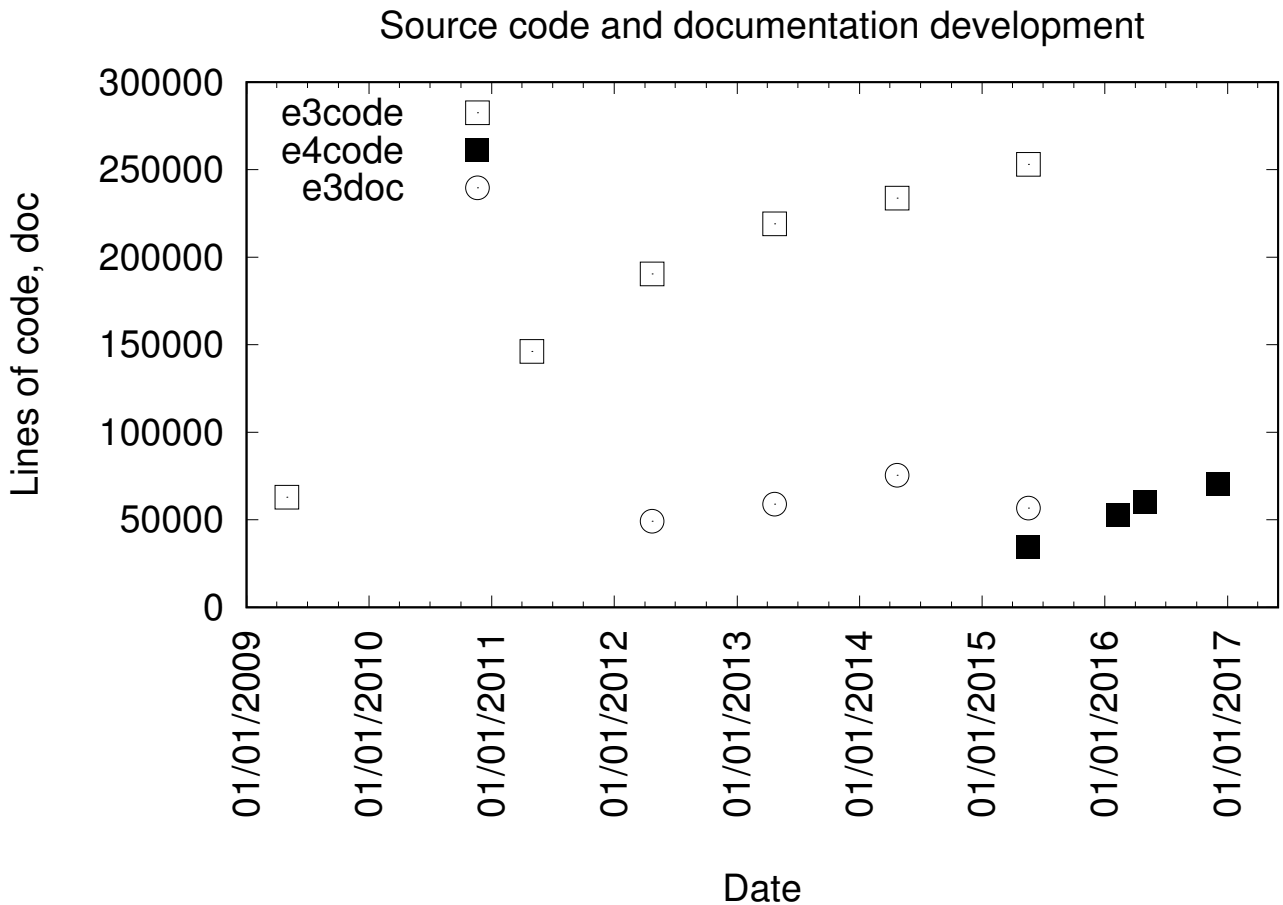
```
1       // First-stage of gas-dynamic update.
2       shared int ftl = 0; // time-level within the overall convective-update
3       shared int gtl = 0; // grid time-level remains at zero for the non-moving grid
4       if (GlobalConfig.apply_bcs_in_parallel) {
5           foreach (blk; parallel(gasBlocks,1)) {
6               if (blk.active) { blk.applyPreReconAction(sim_time, gtl, ftl); }
7           }
8       } else {
9           foreach (blk; gasBlocks) {
10              if (blk.active) { blk.applyPreReconAction(sim_time, gtl, ftl); }
11          }
12      }
13
```

Notes:

- Need to keep most data thread local.
- D Compiler expands “parallel” into code that hands out tasks to the default ThreadPool.

How far have we gone, in lines of source code.

At 60 lines per page,
the Eilmer4 code is equivalent to a 1200 page document.



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.

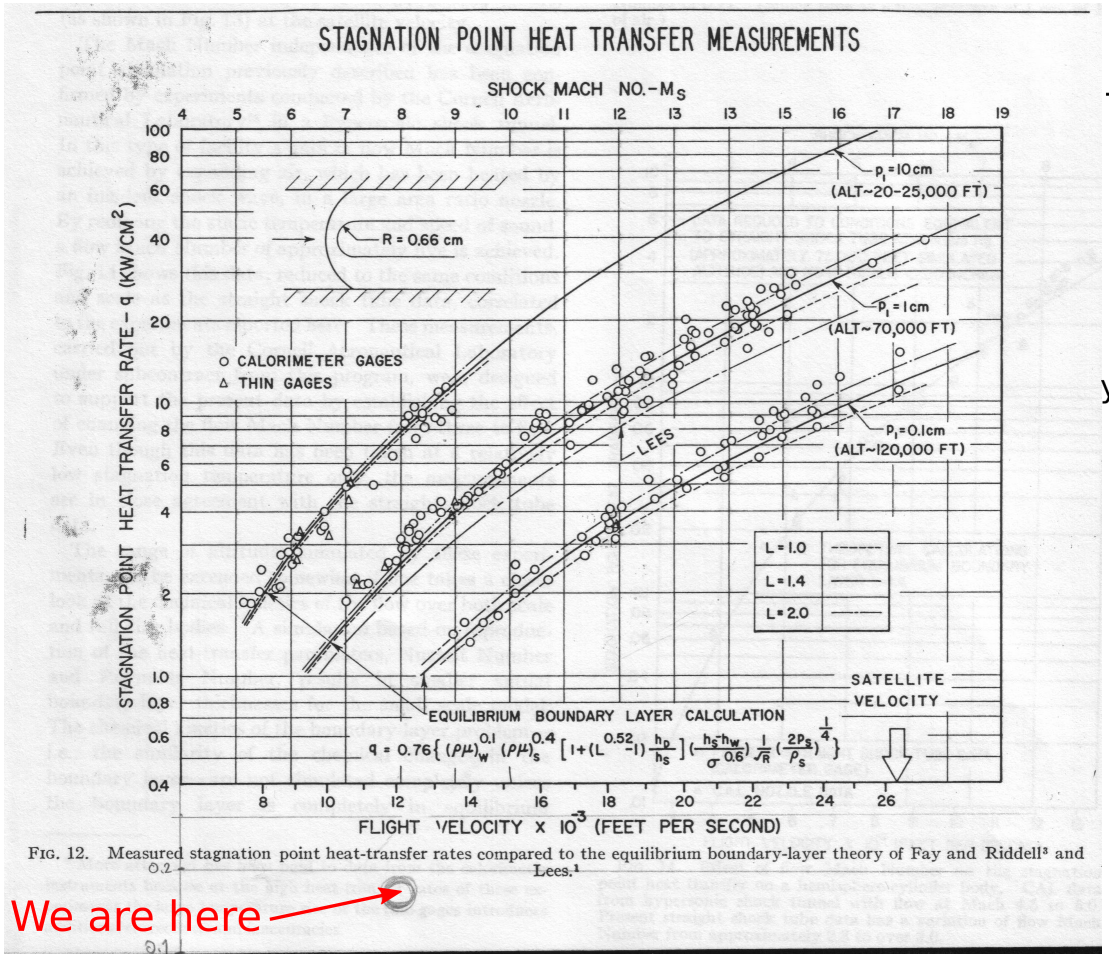
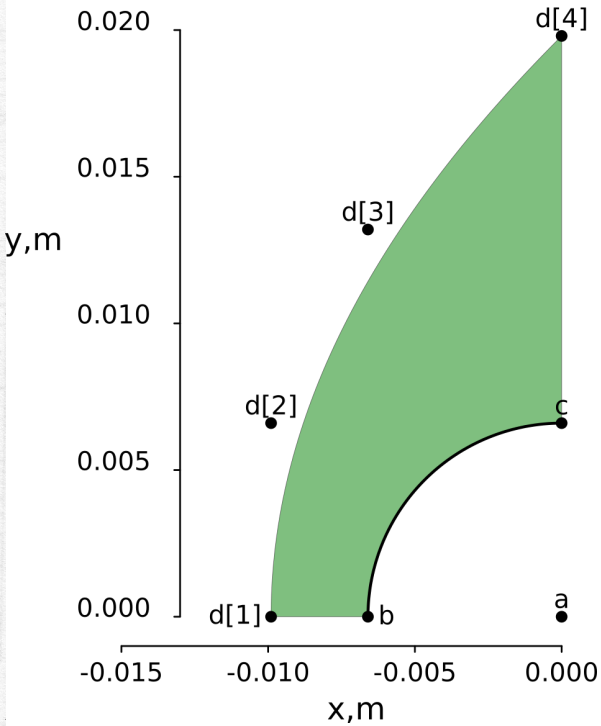


FIG. 12. Measured stagnation point heat-transfer rates compared to the equilibrium boundary-layer theory of Fay and Riddell[3] and Lees.[1]

We are here

flow domain for 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 Ierusalimschy'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

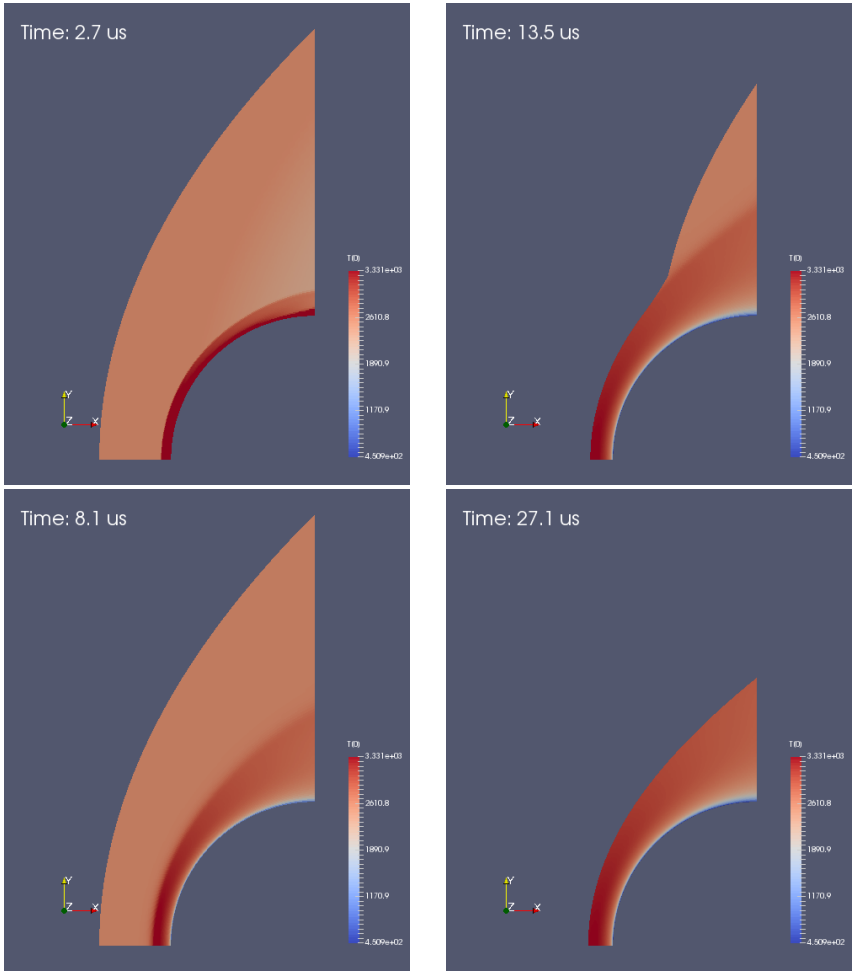
```
blk = SBlockArray{grid=grid, fillCondition=initial, label='blk',
                  bcList={west=InFlowBC_ShockFitting:new{flowCondition=inflow},
                          east=WallBC_NoSlip_FixedT:new{Twall=296.0},
                          north=OutFlowBC_Simple:new{}},
                  nib=1, njb=4}

dofile("sketch-domain.lua")
```

Notes:

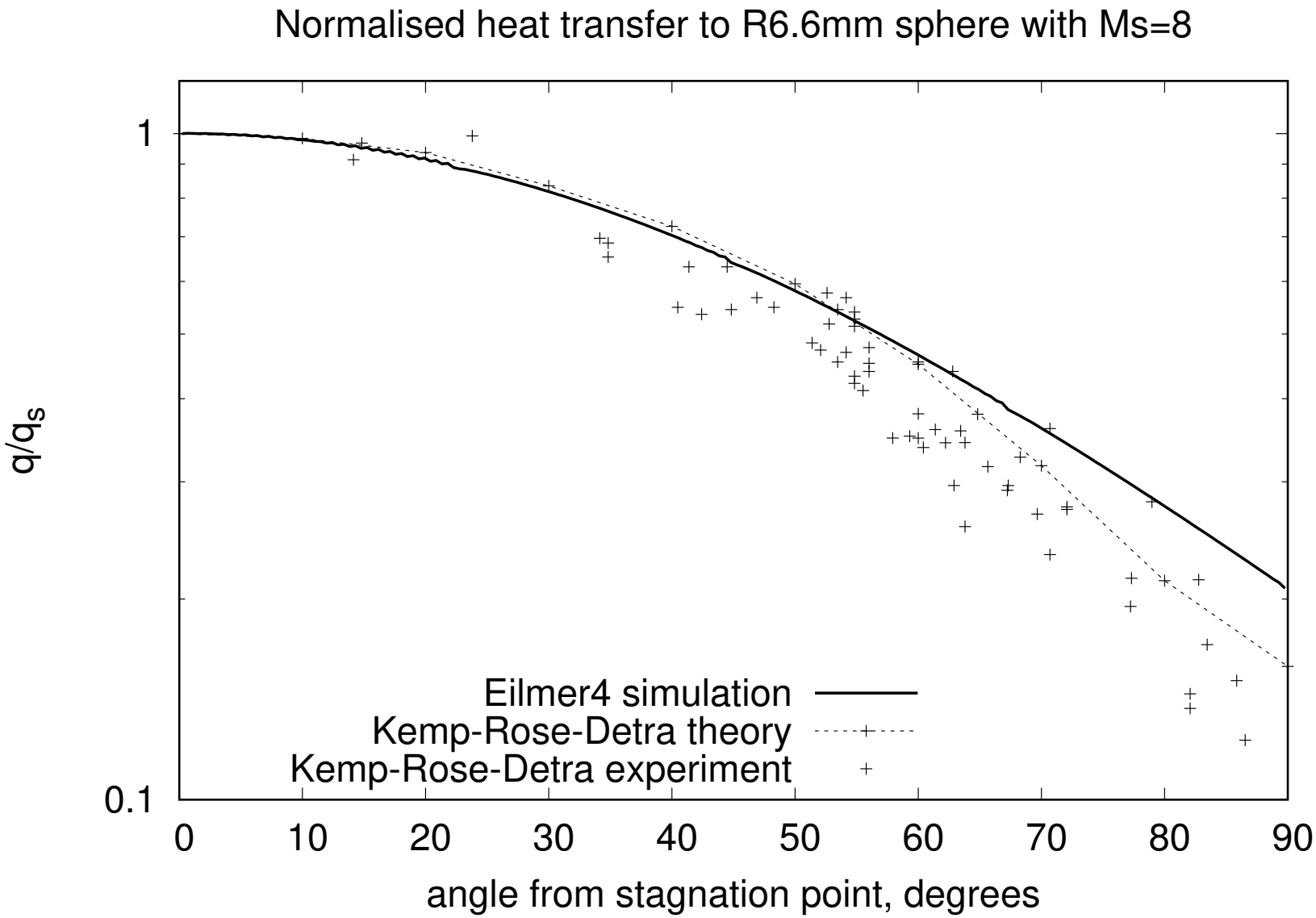
- May define many blocks on a single grid.
- We attach boundary conditions to the domain and specify the initial flow condition.
- Boundary conditions default to class WallBC_WithSlip.
- Some boundary conditions need extra information.

Results – evolution of the temperature field

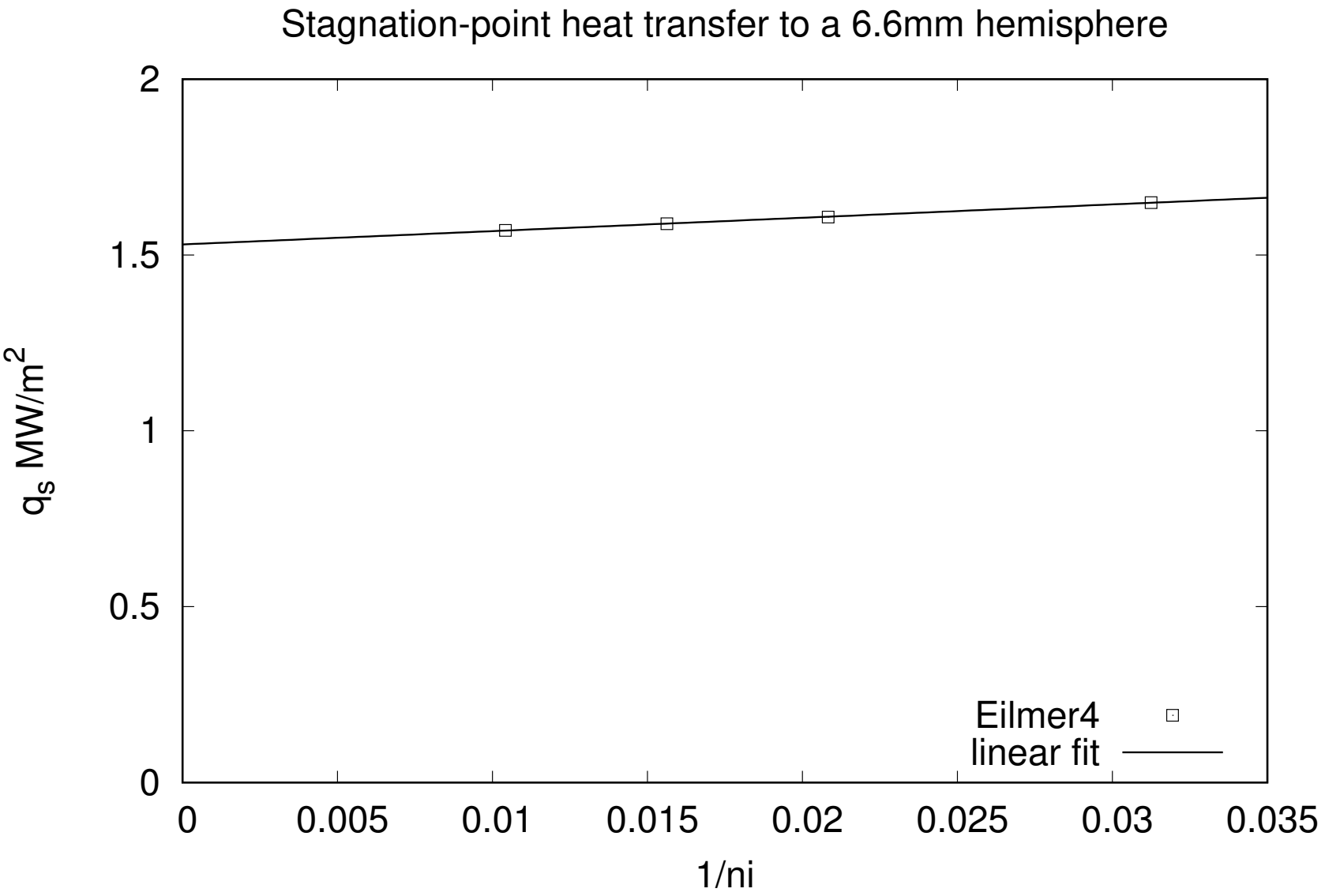


- Characteristic time $\tau = \frac{R}{V_{x,\infty}} = 2.7\,\mu\text{s}$.
- Start viscous effects at 2τ.
- Start shock-fitting at 3τ.
- Run simulation to 10τ.
- Temperature range shown is 451 K to 3331 K.

Results – heat transfer distribution around sphere



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 Czapla, Kyle Damm, Andrew Dann, Andrew Denman, Zac Denman, Luke Doherty, Elise Fahy, Antonia Flocco, Delphine Francois, James Fuata, Nick Gibbons, David Gildfind, Richard Goozeé, Sangdi Gu, Stefan Hess, Jonathan Ho, Carolyn Jacobs, Ingo Jahn, Chris James, Ian Johnston, Ojas Joshi, Xin Kang, Rainer Kirchhartz, Sam Lamboo, Steven Lewis, Pierre Mariotto, 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