MeltPoolDG: FEM-based multi-phase flow solvers for metal additive manufacturing process simulations*

Magdalena Schreter[1,2]

magdalena.schreter@uibk.ac.at, @mschreter

with contributions from: Christoph Meier[2], Peter Munch[2,3], Martin Kronbichler[2,4], Nils Much[2]

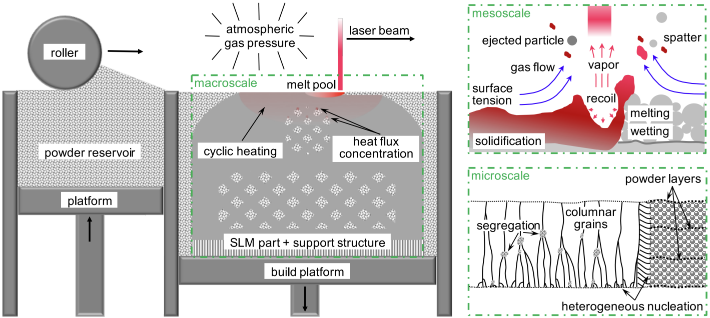
[1]Unit of Strength of Materials and Structural Analysis, Faculty of Engineering Sciences, University of Innsbruck, Austria
[2]Institute for Computational Mechanics, Technical University of Munich, Germany
[3]Helmholtz-Zentrum hereon, Germany
[4]Department of Information Technology, Division of Scientific Computing, Uppsala University, Sweden

Ninth deal.II Users and Developers Workshop, June 18, 2021

*) visit the project on https://github.com/MeltPoolDG/MeltPoolDG

Motivation

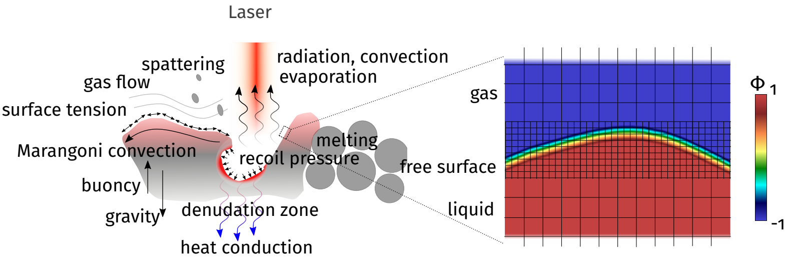
- Powder bed fusion additive manufacturing of metals requires extensive manual process optimization to meet high quality standards
- Physics-based modeling should foster understanding of the governing physics processes to link the process parameters and final part quality
- Multiscale nature $\rightsquigarrow$ individual models to study physical phenomena on length scales needed



Multiscale nature of selective laser melting (taken from Meier C. et al., Annual Review of Heat Transfer (2017))

Scope of the MeltPoolDG project

- **Aim**: development of a predictive model of the melt pool thermo-hydrodynamics including evaporation to provide new insights into crucial physical effects on the mesoscale
- **Challenges**: high density ratios, strong temperature-dependent and interfacial forces, complex interface topologies, complex thermal history, etc.
- **Approach**: (DG)-FEM diffuse-interface-based (level set, phase field) modeling for solving the coupled thermal-hydrodynamical multi-phase flow problem



Overview

1. **MeltPoolDG**
2. **Thermo-hydrodynamics in the melt pool**
3. **Challenge of source term formulation tackled by deal.II features**

MeltPoolDG[3] in a nutshell

- started in summer 2020
- builds upon deal.II[1] and adaflo[2]
- Application-oriented FEM-based solvers for thermo-hydrodynamical problems aiming at mesoscale modeling of additive manufacturing processes
- Easily extendable, modular framework to enable fast implementation of research-driven developments
- Matrix-free and matrix-based solvers
- Adaptive mesh refinement enabling high spatial resolution in interfacial regions
- Simplex support
- Test suite with 60+ test simulations (and a couple of unit tests)

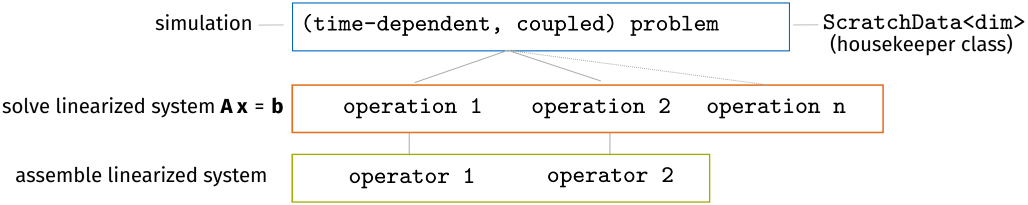
[1] developer version, `https://github.com/dealii/dealii`

[2] M. Kronbichler: `https://github.com/kronbichler/adaflo`

[3] visit us at `https://github.com/MeltPoolDG/MeltPoolDG`

Structure of MeltPoolDG

Three levels of abstraction



Structure of MeltPoolDG

Integration of matrix-free and matrix-based operators

simulation — advection problem — ScratchData<dim> (housekeeper class)

solve linearized system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ — advec operation

```
// matrix-free
if (do_matrix_free)
{
  advec_diff_operator->create_rhs(rhs, solution_old);

  LinearSolve::solve<VectorType,
                     SolverGMRES<VectorType>,
                     OperatorBase<dim, double>>(
                     *advec_diff_operator, solution, rhs);
}
// matrix-based
else
{
  advec_diff_operator->assemble_matrixbased(solution_old,
                                            system_matrix,
                                            rhs);
  LinearSolve::solve<VectorType,
                     SolverGMRES<VectorType>,
                     SparseMatrixType>(
                   system_matrix, solution, rhs);
}
```

assemble linearized system — advec operator

matrix-free

```
void
vmult(VectorType &dst,
    const VectorType &src);

void
create_rhs(VectorType &rhs,
          const VectorType &solution_old);
```

matrix-based

```
void
assemble_matrixbased(const VectorType &solution_old,
                       SparseMatrixType &matrix,
                       VectorType &rhs) const override;
```

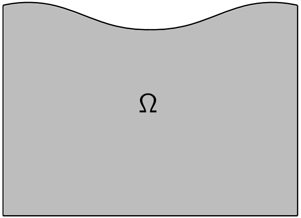
$$\left(w,\ \phi^{n+1}\right)_{\Omega} + \Delta t\left(w,\ \mathbf{u}\cdot\nabla\phi^{n+1}\right)_{\Omega} = \left(w,\ \phi^{n}\right)_{\Omega}$$

Overview

Governing equations

Single-phase flow

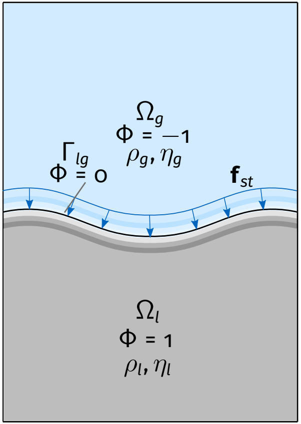
- mass: $\nabla \cdot \mathbf{u} = 0$
- momentum: $\rho \left(\dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} \right) = -\nabla p + \eta \Delta \mathbf{u} + \rho\,\mathbf{g}$



Governing equations

Two-phase flow

- mass: $\nabla \cdot \mathbf{u} = 0$
- momentum: $\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} \right) = -\nabla p + \eta \Delta \mathbf{u} + \rho\, \mathbf{g} + \mathbf{f}_{st}$ surface tension
- level set: $\frac{\partial \phi}{\partial t} + \mathbf{u} \nabla \phi = 0$



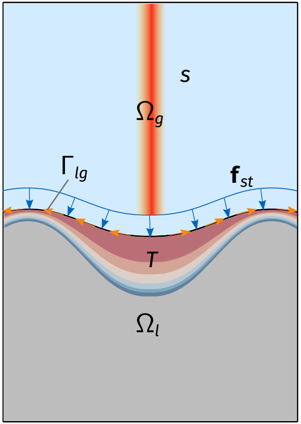
Governing equations

Anisothermal two-phase flow

- mass: $\nabla \cdot \mathbf{u} = 0$
- momentum: $\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\, \mathbf{u} \right) = -\nabla p + \eta \Delta \mathbf{u} + \rho\, \mathbf{g} + \mathbf{f}_{st}$
- level set: $\frac{\partial \phi}{\partial t} + \mathbf{u} \nabla \phi = 0$
- energy: $\frac{\partial \left(\rho\, c_p\, T \right)}{\partial t} + \nabla \cdot \left(\rho\, c_p\, T\, \mathbf{u} \right) = \nabla \cdot (k \nabla T) + s$

temperature-dependent surface tension

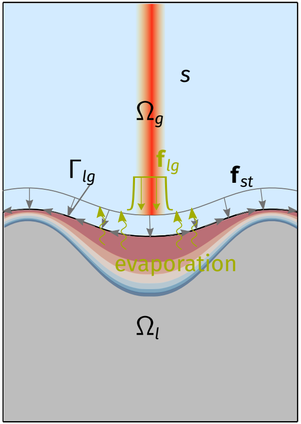
laser heat source



Governing equations

Anisothermal two-phase flow including liquid-vapor phase change

- mass: $\nabla \cdot \mathbf{u} = -\frac{\dot{\rho}}{\rho}$ — evaporative mass flux
- momentum: $\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} \right) = -\nabla p + \eta \Delta \mathbf{u} + \rho\,\mathbf{g} + \mathbf{f}_{st} + \mathbf{f}_{lg}$ — recoil pres- sure
- level set: $\frac{\partial \phi}{\partial t} + \mathbf{u}_\Gamma \nabla \phi = 0$
- energy: $\frac{\partial (\rho\, c_p\, T)}{\partial t} + \nabla \cdot (\rho\, c_p\, T\, \mathbf{u}) = \nabla \cdot (k \nabla T) + s + s_{lg}$ — evaporative heat flux



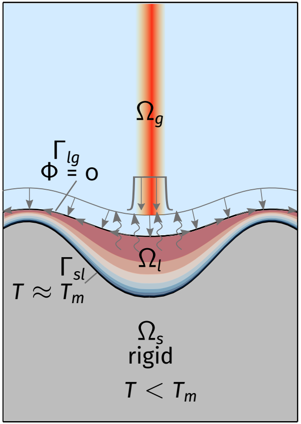
Governing equations

Melt front propagation

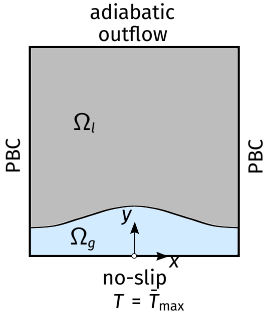
We distinguish between

- gaseous domain $\Omega_g = \{\mathbf{x} \mid \Phi(\mathbf{x}) \leq 0\}$
- liquid domain $\Omega_l = \{\mathbf{x} \mid \Phi(\mathbf{x}) > 0; T(\mathbf{x}) \geq T_m\}$
- solid domain $\Omega_s = \{\mathbf{x} \mid \Phi(\mathbf{x}) > 0; T(\mathbf{x}) < T_m\}$

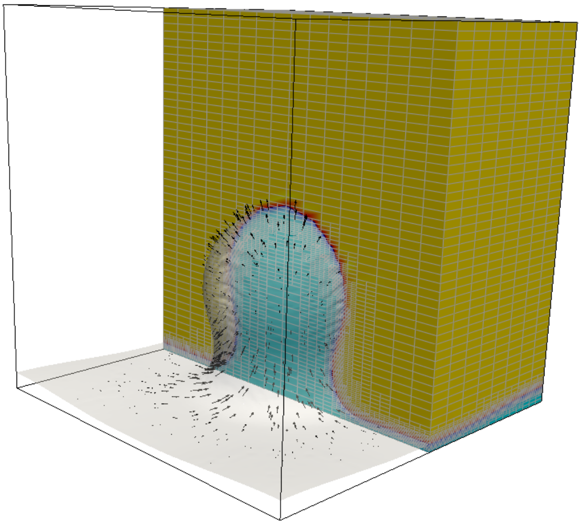
Note: The two-phase flow is solved in $\Omega_g \cup \Omega_l$, while the heat transfer is solved in $\Omega = \Omega_g \cup \Omega_l \cup \Omega_s$.



Example 1: film boiling[4]



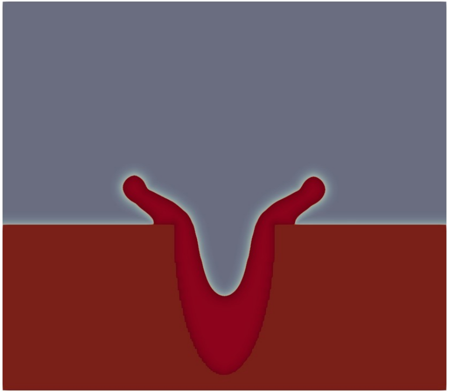
$$\Omega = x, y \in [-0.04, 0.04] \times [0, 0.08]$$



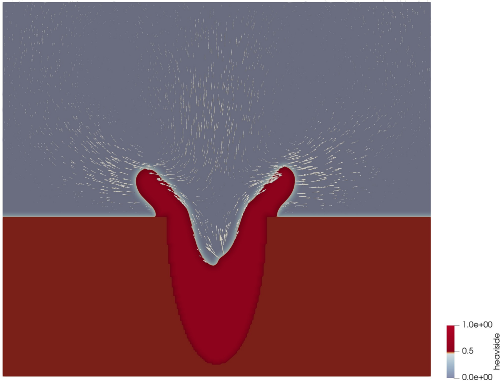
[4] Hardt & Wondra, J Comp Physics (2008); Gibou et al., J Comp Physics (2006)

Example 2: melting of a metal plate[5]

empirical recoil pressure force (no mass flux)



consideration of evaporative mass flux



[5] C. Meier et al. (2021): submission to *GAMM Mitteilungen*

Overview

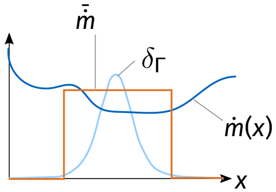
1. MeltPoolDG
2. Thermo-hydrodynamics in the melt pool
3. Challenge of source term formulation tackled by deal.II features

Challenge of source term formulation tackled by deal.II features

Problem: Departing from our diffuse level-set based interface formulation, we would like to compute a field variable holding *averaged* values across the interface.

Goal: Perform a weighted line integral along the normal of the interfacial domain.

$$\bar{\dot{m}} := \int_x \dot{m}\delta_\Gamma dx$$



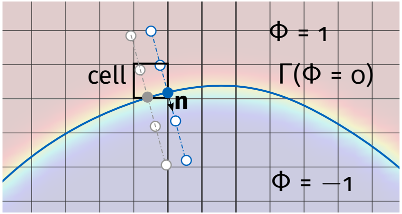
Question: Is it possible using `deal.II` utilities?

Perform a weighted line integral of values along the interface normal

Workflow:

- Generate a cloud of points being normal to the interface

```
GridTools::MarchingCubeAlgorithm<dim, VectorType>
mc(mapping, dof_handler.get_fe(), n_subdivisions);
// compute the vertices at the interface cellwise
mc.process_cell(cell, level_set_vector, 0, interface_vertices,
      interface_cells);
// evaluate the normal vector at arbitrary points within the cell
FEPointEvaluation<dim, dim> phi_normal(mapping, fe_normal, update_values);
phi_normal.reinit(cell, unit_points);
cell->get_dof_values(normal_vector, buffer); // get the nodal values ...
// ... and interpolate them to the unit_points
phi_normal.evaluate(buffer, EvaluationFlags::values)
phi_normal.get_value(unit_point_idx);
// store the point cloud
std::vector<Point<dim>> points_normal_to_interface = /*...*/;
```



Perform a weighted line integral of values along the interface normal

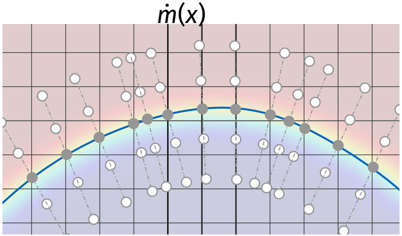
Workflow:

- Generate a cloud of points being normal to the interface
- Gather the results at the point cloud from a DoF-vector

```
// evaluate at point cloud
Utilities::MPI::RemotePointEvaluation<dim, dim>
remote_point_evaluation(1e-6 /*tolerance*/, true /*unique mapping*/);

remote_point_evaluation.reinit(points_normal_to_interface,
                               triangulation,
                               mapping);

const auto mass_flux_point_cloud =
VectorTools::point_values<1 /*n_components*/>(remote_point_evaluation,
                                              dof_handler
                                              mass_flux);
```



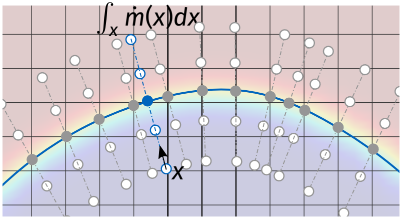
Perform a weighted line integral of values along the interface normal

Workflow:

- Generate a cloud of points being normal to the interface
- Gather the results at the point cloud from a DoF-vector
- Perform a weighted line integral and set the point cloud values equal to the latter

```
for (auto i = 0u; i < points_normal_to_interface_pointer.size() - 1; ++i)
{
  const auto start = points_normal_to_interface_pointer[i];
  const auto size  = points_normal_to_interface_pointer[i + 1] - start;

  double line_integral = 0;
  // loop over all points along normal at one MC point
  for (unsigned int l = 0; l < size; ++l)
    line_integral += integration_weight * mass_flux_point_cloud[start+l]
  // overwrite values with averaged one
  for (unsigned int l = 0; l < mass_flux_evaluation_values.size(); ++l)
    mass_flux_point_cloud[start+l] = line_integral;
}
```



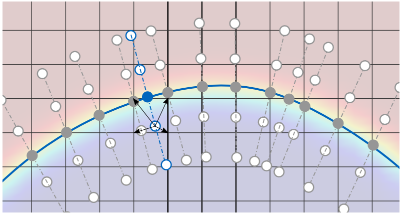
Perform a weighted line integral of values along the interface normal

Workflow:

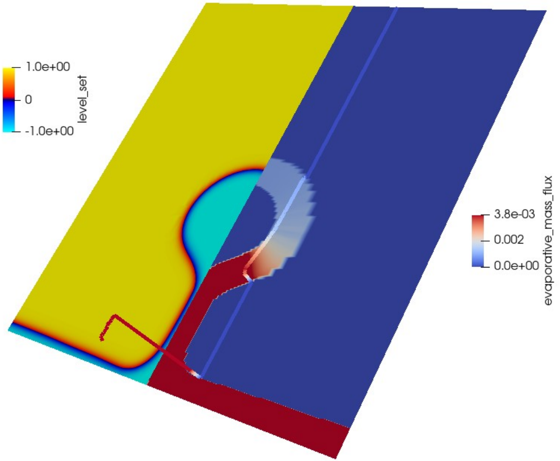
- Generate a cloud of points being normal to the interface
- Gather the results at the point cloud from a DoF-vector
- Perform a weighted line integral and set the point cloud values equal to the latter
- Broadcast values from cloud points along normal to the nodal points

```
VectorType solution;
const auto broadcast_function = [&](const auto &values, const auto &
      cell_data) {
  /* broadcast values from interior (values) to nodal points and fill
        Vector solution */
};
std::vector<double> buffer;
remote_point_evaluation.template process_and_evaluate<double>(
  mass_flux_val,
  buffer,
  broadcast_function);

solution.compress(VectorOperation::add);
```

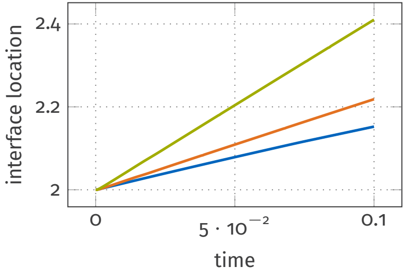


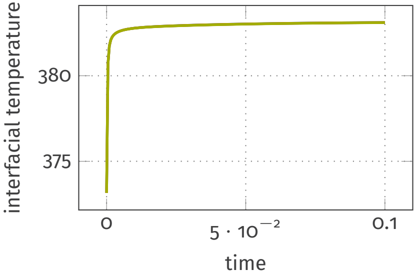
Benchmark example: film boiling



Conclusions

- `MeltPoolDG` is an **application-driven research code** providing solvers for simulating the thermo-hydrodynamics in the vicinity of the melt pool based on `deal.II` and `adaflo`
- Focus on **physics-based modeling of melt pool processes**, being a key model component for the holistic modeling of metal additive process simulations
- Apart from the existing `deal.II` features, **novel features offer new possibilities for evaluations in numerical computations** and are also helpful for **postprocessing** purposes





Ongoing projects and outlook

Physics:

- Phase-field methods (to account for multi-component systems including phase changes)
- Consideration of metal vapor as an individual phase
- Raytracing schemes
- Mobile, deformable particles

Numerical schemes:

- Spatial discretization by means of Discontinuous-Galerkin-FEM
- More sophisticated schemes for fluid-structure interaction at the solid-liquid/solid-gaseous interface (e.g. immersed boundary method)
- Coupling between DEM (solid phase) and FEM (liquid phases) including phase change
- Narrow-band schemes/non-matching grids

MeltPoolDG: FEM-based multi-phase flow solvers for metal additive manufacturing process simulations*

Magdalena Schreter[1,2]

magdalena.schreter@uibk.ac.at, @mschreter

with contributions from: Christoph Meier[2], Peter Munch[2,3], Martin Kronbichler[2,4], Nils Much[2]

[1]Unit of Strength of Materials and Structural Analysis, Faculty of Engineering Sciences, University of Innsbruck, Austria
[2]Institute for Computational Mechanics, Technical University of Munich, Germany
[3]Helmholtz-Zentrum hereon, Germany
[4]Department of Information Technology, Division of Scientific Computing, Uppsala University, Sweden

Ninth deal.II Users and Developers Workshop, June 18, 2021

---

*) visit the project on https://github.com/MeltPoolDG/MeltPoolDG