0.1 FLASH API

0.1.1 Client Functions (FrontTracking Module)

All functions listed in this section are part of the Front Tracking module, designed for the FLASH source tree. They are all named FrontTracking_* where * is replaced with the function name below. This follows the FLASH naming convention for module functions.

0.1.1.1 init

Summary: A simple interfacing function for the Front Tracking initialization routine. The initialization step consists of passing the required Grid information to the Front Tracking library so that it can properly initialize the front. A call to the propagate routine with a dt of 0.0 is needed to allow the front to simplify the surface that may have been created during an initialization. In addition, a call to initialize the component array is invoked.

Inputs: None

Outputs: None

Library Functions Called:

- FTAPI_init

0.1.1.2 updateComp

Summary: For efficiency, we maintain a data structure to store the components (material IDs) at each cell center in the domain. Thus the **updateComp** function is needed to update the components in each cell (if necessary) after the front is propagated.

When a cell center is passed by a moving front, the component changes in the process. Since the physical state stored at that cell center is supposed to be representative of the component of the cell, the state needs to be updated to reflect its new component. To update the state in a physically consistent way, we solve a 1D Riemann problem in the direction of the front normal across the cell center. The direction of the front normal is found via interpolation from normals on the front, using front points nearby to the cell. The left and right input states for the Riemann problem are taken as the old state of the cell, prior to front movement and the component change, and the cell's ghost state, as defined in the **fillGhostStates** function, for the material it is becoming.

The contact wave solution p^* and u^* (converted back to grid velocities using the interpolated normal vector and tangential velocity of the ghost front state) are the new state pressure and velocity. The density for the new state is determined in a thermodynamically consistent way, using a wave curve expansion based on the difference in pressure between p^* and the ghost front state pressure.

Inputs:

- Ghost State Array calculated in **fillGhostStates**

Outputs:

- Component Array

Library Function Called:

- FTAPI_getComponent (material ID at a coordinate point)
- FTAPI_gridNormal (front normal vector interpolated to a grid point)

0.1.1.3 propagate

Summary: A simple interfacing function for the Front Tracking propagation routine. The propagation step consists of four main parts:

1. FillGhostStates,
2. UpdateGuardCells,
3. Propagate and
4. UpdateComp

. Updating guard cells is needed to allow for the ghost states set in the first step to be applied to any buffer cells, using the appropriate boundary conditions. The Propagate step, calls the API library function that invokes the propagation routine.

Included in the propagate routine is a subcycling loop, which can be needed if the front dt restriction is more severe than that calculated by FLASH. This is a rare occurrence, but can happen if the velocity at the front point is determined to be larger than those on the grid. An alternate option would be to build in an interface to allow the front dt and FLASH's dt to both be taken into account when the step dt is calculated, however that has not been implemented in this version. Instead, the front will propagate however many times are necessary to allow it to reach the same elapsed physical time as that of the underlying FLASH physics step. As long as the components and ghost states are updated in tandem with each propagation, this will be physically consistent during the subcycling steps.

Inputs:

- *dt* of the current timestep

Outputs: None

Library Functions Called:

- FTAPI_propagate

0.1.1.4 getVel

Summary: The **getVel** function is designed to fulfill a request by the front for the velocity of a front point (i.e. a location in the domain) so that it can be propagated. The front represents the Lagrangian propagation of an initial isoconcentration surface between two fluids. To be physically consistent in the Lagrangian propagation a Riemann problem is solved between a representative state for the materials which we refer to as a left state and right state. To determine the left and right states at the front point, we use a procedure adapted from Bo et. al [1], which reconstructs front states on the fly from an interpolation of a bounding box of cells relative to the front point. On the side of the interface where these are physical (real) states, the real state is taken and on sides where the other material is present, a ghost fluid state, obtained following the procedure outlined in the **fillGhostStates** function is used.

The velocities for the left and right front states are then decomposed into a velocity normal to the front and tangential to the front, by dotting with a normal vector for the front point of interest. This produces a 1-dimensional input state along the normal direction of the front, using the normal velocity.

The resulting 1-dimensional Riemann problem is solved, here using the built-in Riemann solver in FLASH (using the method of Collela and Glaz [3]) and the Riemann solution contact wave velocity, u^* is the desired front point's velocity. Note that since the front is only propagated in the normal direction [1, 2], this is the velocity to be returned back through the API to allow the Front Tracking library to move the points.

Inputs:

- Coordinates of the front point
- Left side component (material ID) at the front point
- Right side component (material ID) at the front point
- Ghost State Array calculated in **fillGhostStates**

Outputs:

- Velocity of the front point

Library Functions Called:

- FTAPI_normal (normal vector at a front point)

0.1.1.5 fillGhostStates

Summary: This function is a helper function in that it is not required to implement the front tracking API, but helps organize and simplify the implementation. In the **getVel** function, the velocity of a front point is obtained from the solution of a Riemann problem. The inputs to this Riemann problem are a representative state for each component. To obtain the representative state for component i, we interpolate from the nearby cells which

match that component ID. The bounding box of cells centers for the front point will almost certainly not be all of the same component, as the interface surface must be between the cells. Thus, instead of trying to handle interpolation of each case accordingly, a two-step interpolation procedure is employed.

In the first step, which is carried out here in the **fillGhostStates** function, all cells nearby to the front calculate a double-valued state function, one for each component ID. For the component ID that matches the component of the cell being acted on, the physical state is copied into the ghost state. For the other component ID, the one that does not match the physical component in the cell, an average state is obtained from its neighbors with a physical component matching it. For example, for a cell which has physical component 2, GhostState2 would be set equal to the state in the current cell and GhostState1 would be set to be an average of the neighbors to the current cell which have physical component 1.

The definition of a neighbor is somewhat arbitrary and depends on the underlying structure of the mesh. For our implementation in FLASH, which uses a fixed, structured mesh, we define the average state from the closest bounding box of cells which contains a cell with the desired physical component.

With the double-valued state function at every cell near the front, the second stage of the interpolation, carried out as part of the **getVel** function can be carried out using a standard bi-linear interpolation from a set of like component ghost states. The name fillGhostStates may be somewhat of a misnomer, in that they don't represent the ghost state as defined in Bo et. al [1], but rather an average state, used to aid in the process of calculating the front states.

The ghost states defined here are also used in the **updateComp** function, as the input to the Riemann problem used to determine the correct state for a cell which has had its component changed due to front propagation.

Inputs: None

Outputs:

- Ghost State Array

Library Functions Called: None

0.1.1.6 output

Summary: A simple interfacing function for the Front Tracking output routine. The output step consists of passing a request to the front to dump either a checkpoint file or a plot file.

Inputs:

- Flag for checkpoint or plotFile

Outputs: None

Library Functions Called:

- FTAPI_output
- FTAPI_writeRestart

0.1.2 Modified FLASH functions

In addition to developing a set of client functions to allow FLASH to interface with the FTAPI, modifications to existing FLASH files were needed to fully implement the front tracking algorithm. Front propagation and consistency of the front surface is all handled on the FTAPI side, but the other half of front tracking, which is the modification of stencils during the interior update, needs to be directly applied to the interior solvers implemented in the client code. The functions that were modified and a description of those changes are listed here.

0.1.2.1 hy_ppm_block

In the current version of the FLASH FTAPI, the full active front tracking algorithm is only implemented for the split PPM hydro solver. In an actively tracked algorithm, interior stencils which contain states from both components are modified so that when solving a cell for a particular component ID, only states which match that component ID are used in the stencil. Since those states are not always physical, i.e. when the stencil takes you across the front, they are replaced with a ghost cell state, determined from a Riemann solution between the physical left and right states at the front. This Riemann solution provides the pressure and velocity, the density is determined using a wave curve expansion for the particular component. The reader is referred to the seminal front tracking paper [2] for a full discussion of this algorithm.

In **hy_ppm_block** the stencils are set up for the interior solve, so this is the natural place to make adjustments to the stencil. Once the stencil has been corrected, the PPM algorithm can proceed as designed. To correct the stencil, the sweep is carried out from left to right only up to the point that the component changes. Once a component change is encountered (i.e. a front point has been crossed), a Riemann solution between the two distinct component cells bounding the front point is solved. This Riemann solution and its corresponding midstate density for the component being solved is then used to populate the rest of the stencil. This corrects the stencil and the PPM algorithm is allowed to proceed from there. After a call to **hydro_1d**, fluxes for each cell face are stored for use in the **hy_ppm_updateSoln** function. As these fluxes are only valid for the cells which had the primary component ID, those fluxes are stored and the rest are discarded. The sweep then continues starting from where it left off and working on this new component until a front point is encountered. Again, a Riemann solve is used to populate the remaining stencil on both the left and right sides and fluxes for only the valid component cells are stored. This procedure continues until the entire sweep has finished.

At cell faces where the two neighboring cells have different components, two fluxes, one for each component is calculated. A second array is created to flag the faces where the flux function is dual-valued and store the alternate flux for use during the update to the state variables carried out in the **hy_ppm_updateSoln** function. Having a double-valued flux

function at these faces creates a small error in conservation, which is the focus of ongoing work [6, 8], and has been a known minor drawback of the current front tracking algorithm. However, a long and extensive track record of verification and validation [5, 7, 4] suggests that this should not have a meaningful impact on the current results.

0.1.2.2 hy_ppm_updateSoln

As detailed in the above section, the flux function is dual valued at cell faces which border two cells with different components. Only a small modification is carried out here identifying when we have a face with a dual valued flux function and to choose the correct one. This is done by consistently making the flux for the cell which borders to the left the one that is stored in the alternate flux array.

0.1.2.3 rieman

The Riemann solver in FLASH uses the method of Collela and Glaz [3] and is configured to run directly with the hydro unit over a specified 1-dimensional region in the domain, taking a vector of inputs and producing a vector of outputs. For our added uses of the Riemann solver for front tracking we only have a single Riemann solution that is needed. A duplicate of this function was created and configured to take a single left and right state input vector and produce a single output solution without the use of additional step through functions to produce left and right extrapolated states.

In addition, instead of returning an average density, the function returns two densities, one for the left side contact and one for the right side contact from the Riemann wave expansion. To determine the correct densities for the left and right solutions, the input pressures are compared to the Riemann solution pressure, p^*. If $p_I > p^*$ the wave is a shock and if $p_I < p^*$ the wave is a rarefaction, where p_I is the respective input pressure for that side. Once the wave type is determined, a wave curve expansion can be calculated to determine the thermodynamically consistent density for each side of the contact wave.

0.1.2.4 Additional functions

Additional minor changes are made in **hy_ppm_interface** and **hy_ppm_sweep** to allow for the additional flux array structure. Since the major changes to the FLASH code impact only the sweep algorithm, only minor changes, mainly for the purposes of invoking the Front Tracking module functions described in Sec. 0.1.1, were also needed. A change was made to **Driver_evolveFlash** to add calls to **FrontTracking_propagate** at the beginning of each physics sequence. In addition, **Driver_initFlash** and **IO_output** were modified to add a call to the FrontTracking module's **FrontTracking_init** and **FrontTracking_output**, respectively.

0.1.3 API Testing and Proof of Concept

Since the API consists of a major implementation upgrade to FLASH, a few tests were carried out to verify that the code was correctly solving the fluid equations and that there were no bugs present in the implementation. Since both FLASH and FronTier have been

Table 1: Initial conditions for the Sod shock tube problem.

	Right	Left
Density	1	0.125
Pressure	1	0.1
Velocity	0	0

heavily verified and validated, simple A/B comparisons between FLASH with and without the front tracking algorithm turned on can provide a solid testing suite as well as allow for a proof of concept for the advantage of front tracking. We detail a few of the tests conducted here and discuss the advantages of the use of front tracking.

0.1.3.1 Sod Shock Tube

One of the first tests that any fluid code should be subjected to is the shock tube problem described by Gary Sod [9]. This is a basic test of a fluid solver modeled after a shock tube, where a density and pressure discontinuity is initialized in the center of the domain producing a shock wave, contact wave and rarefaction wave which propagate through the domain as time evolves. The shock tube problem has the benefit of an analytic solution to compare to [9]. The initial conditions for our problem follow those laid out in the FLASH manual and detailed in Table 1 where the domain is setup from $x = 0$ to 1.

The simulations are all run in 2D with a single symmetric interface at $x = 0.5$. This simplifies to a 1D physics computational model. In Fig. 1 we plot both the FLASH base simulation (blue, dotted) and the FLASH simulation with front tracking coupled in (red, dot-dash), along with the exact solution (black, solid). The resolution for the FLASH runs presented here is 250 grid cells. The exact solution is calculated at 1000 evenly spaced grid points, using the exact Riemann code found at `http://cococubed.asu.edu/code_pages/exact_Riemann.shtml`. In the left frame, we plot the full domain. Some differences, as expected between the exact solution and the FLASH solutions are evident at the shock and contact discontinuities. The shock, which is not tracked by FT, shows almost no differences between the two FLASH simulations, while the contact shows a much sharper interface captured by the FT version of FLASH relative to the base case. To emphasize these differences, the contact discontinuity is blown up in the right frame. The FLASH version with FT does a much better job of capturing the sharpness of the contact relative to the base simulation, even at this resolution of 250 cells.

For a quantitative analysis, we conduct a convergence study for both simulations ranging over resolutions between 50 cells and 1600 cells. In Table 2 we plot the average relative pointwise error in density for each simulation relative to the exact solution. The average relative L1 error is calculated in a pointwise fashion, comparing each grid cell value in the simulation to the exact solution at that point (see Eq. 1).

$$\overline{\text{Error}_{\text{L1}}} = \frac{1}{N} \sum_{i \in grid}^{N} \frac{|\text{sim}_\text{i} - \text{exact}_\text{i}|}{\text{exact}_\text{i}} \tag{1}$$

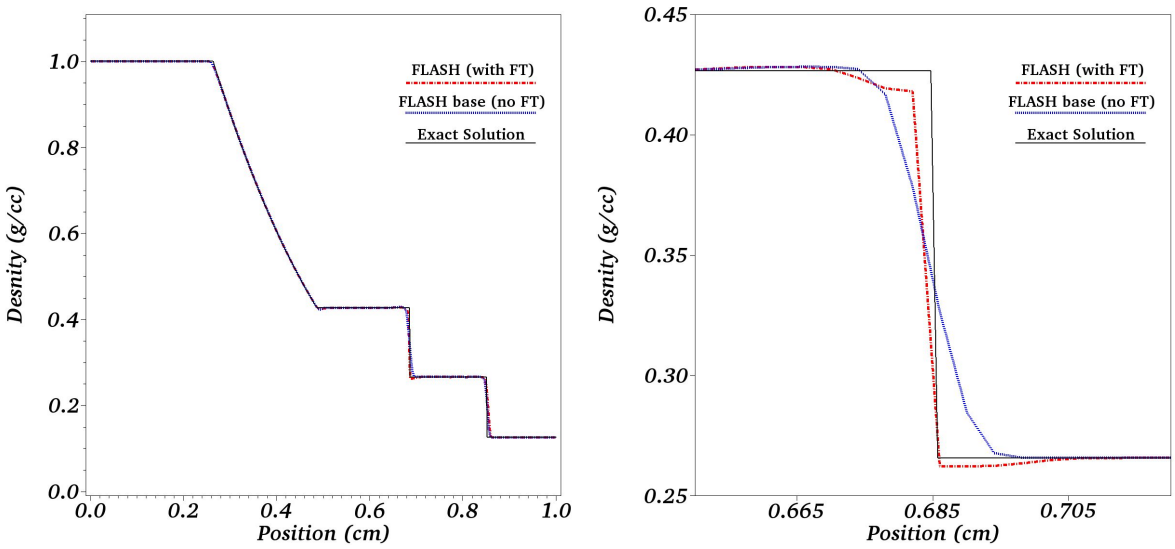

Figure 1: Sod shock tube simulation at t=0.2 using 250 grid cells in the x direction. The base FLASH simulation is the blue dotted line, the FLASH simulation with FT turned on is the red dot-dash line and the exact solution is the black solid line. Left: full domain and Right: zoomed in picture of the contact discontinuity, where front tracking has the major impact.

Table 2 shows that at the coarsest resolution, front tracking has about a 15% improvement over the base FLASH simulation and that as the resolution increases, the improvement decreases but for the most part is still maintained. As this comparison is taken over the whole domain and front tracking only has a major impact on the small region around the contact, we conduct the same analysis, calculating the relative L1 error only over the region in space between $x = 0.65$ and $x = 0.72$, which brackets the contact discontinuity which occurs at 0.685 in the exact solution. A drastic improvement in the average L1 error is observed in the contact region where front tracking has the main effect. The FT simulation improves the average error by approximately 5 times throughout all resolution levels tested here. We also calculate the convergence orders in both Tables 2 and 3. The convergence order $p = log_2(E_c/E_f)$, where E_c and E_f represent the errors on consecutive grid levels with c the coarser grid and f the finer grid. The log_2 in the convergence order calculation is valid for resolution increases of a factor of 2. Over the full domain (Table 2), with the exception of the first comparison, the convergence order is roughly the same between the two simulations with some variability between them. For just the region near the contact discontinuity (Table 2), the FT simulation shows an improvement of nearly a 1st order convergence across all domain level, again with the exception of the 50 to 100 comparison. This is an improvement over the base FLASH simulation which has a dropping convergence order over the majority of the comparisons and is shy of 1 on all grids except for the first two. A convergence order of 1, despite using a PPM method in FLASH which is 3rd order is to be expected for the shock tube problem as the presence of shocks and discontinuities tend to prevent the observance of higher than 1st order convergence.

Table 2: Relative L1 errors for the entire domain between the FLASH base simulation and FT simulation relative to the exact solution. Resolution ranges from 50 cells to 1600 cells. The FT simulation seems to show a slight advantage over the base FLASH simulation.

Grid Resolution	Relative L1 error		Convergence Order	
	Base FLASH	FLASH with FT	Base FLASH	FLASH with FT
50	0.0183	0.0152	-	-
100	0.0108	0.0111	0.76	0.45
200	0.0057	0.0056	0.92	0.99
400	0.0031	0.0030	0.88	0.90
800	0.0019	0.0017	0.71	0.82
1600	0.0008	0.0005	1.25	1.77

Table 3: Relative L1 errors for the region bracketing the contact discontinuity. Resolution ranges from 50 cells to 1600 cells. The FT simulation has an average relative error around 5 times better than the base FLASH simulation for this region.

Grid Resolution	Relative L1 error		Convergence Order	
	Base FLASH	FLASH with FT	Base FLASH	FLASH with FT
50	0.1081	0.0201	-	-
100	0.0541	0.0145	1.00	0.47
200	0.0294	0.0076	0.88	0.93
400	0.0173	0.0038	0.77	1.00
800	0.0104	0.0019	0.73	1.00
1600	0.0055	0.0009	0.92	1.08

0.1.3.2 Rayleigh-Taylor Single Mode

We conduct a 2D single mode RT simulation with periodic boundary conditions over a domain ranging from -0.25 to 0.25 in x and reflecting walls over a range of -0.75 to 0.75 in y. The initial perturbation is a velocity perturbation following the setup provide at `http://www.astro.princeton.edu/~jstone/Athena/tests/rt/rt.html` with $v_y = 0.0025(1 + \cos(4\pi x)(1 + \cos(3\pi y))$. The interface is at $x = 0$ with an upper density of $\rho_H = 2$ and lower density of $\rho_L = 1$. The pressures are initialized to maintain hydrostatic equilibrium with $P = 2.5 - 0.1 * \rho y$ with gravity = -0.1. The adiabatic index $\gamma = 1.4$. The resolution is 50 by 250 cells.

In Fig. 2 we plot the FLASH simulations with (right) and without (left) front tracking coupled in through the API. The plot is shown at a time corresponding to $t = 10$s, well into the non-linear regime. Both simulations show similar macro structures relative to growth of the instability and shape suggesting the API has been installed successfully. Some clear advantages of front tracking are observable when comparing the two simulaitons. The front tracking simulation has no numerical diffusion and is overlayed by the tracked front. Additional small scale details are visible in the front tracked solution, which as a result of the numerical diffusion in the base FLASH simulation are not resolved. Also, the front tracked solution has a better shape, a different length scale for the bubbles and spikes and eliminates the dip observed at the top of the bubble. These highlight some of the key reasons front tracking has been able to perform better with regard to RM and RT instability modeling.

0.1.3.3 Rayleigh-Taylor Multi Mode

Here we analyze A to B comparisons of a 2D multi mode RT simulation with an initial random perturbation. The domain is a square box with both x and y ranging from -0.25 cm to 0.25 cm. The configuration is similar to the single mode RT problem described above with densities of 1 g/cc and 2 g/cc, hydrostatic pressures and a gravity of -0.1 cm/s^2. The boundary conditions are periodic in x and reflecting in y with $\gamma = 1.4$.

The initial perturbation consists of random amplitudes, phases and mode numbers selected from a mode number range of 8 to 30. The amplitudes are scaled to produce a maximum initial perturbation amplitude of 0.003 cm. Since the smallest wavelength λ is 0.0167 cm, the amplitude is $\approx 0.2\lambda$ or starting just outside of the linear regime. The simulations are run with a grid resolution of 1000^2, providing ≈ 33 cells of resolution for the smallest initial wavelength.

In Fig. 3 we plot the FLASH simulation with (bottom) and without (top) front tracking coupled in. The plot is shown at $t = 2.75$s at which point bubble competition is beginning to start but the structures are still relatively coherent. It is immediately apparent one of the major advantages of front tracking, as the top frame contains quite a bit of numerical diffusion acting at the interface between the two fluids.

Front tracking prevents this numerical diffusion and thus provides a role for a physical diffusion model to be coupled in to allow for physically consistent length scales associated with the diffusion processes in the problem of interest.

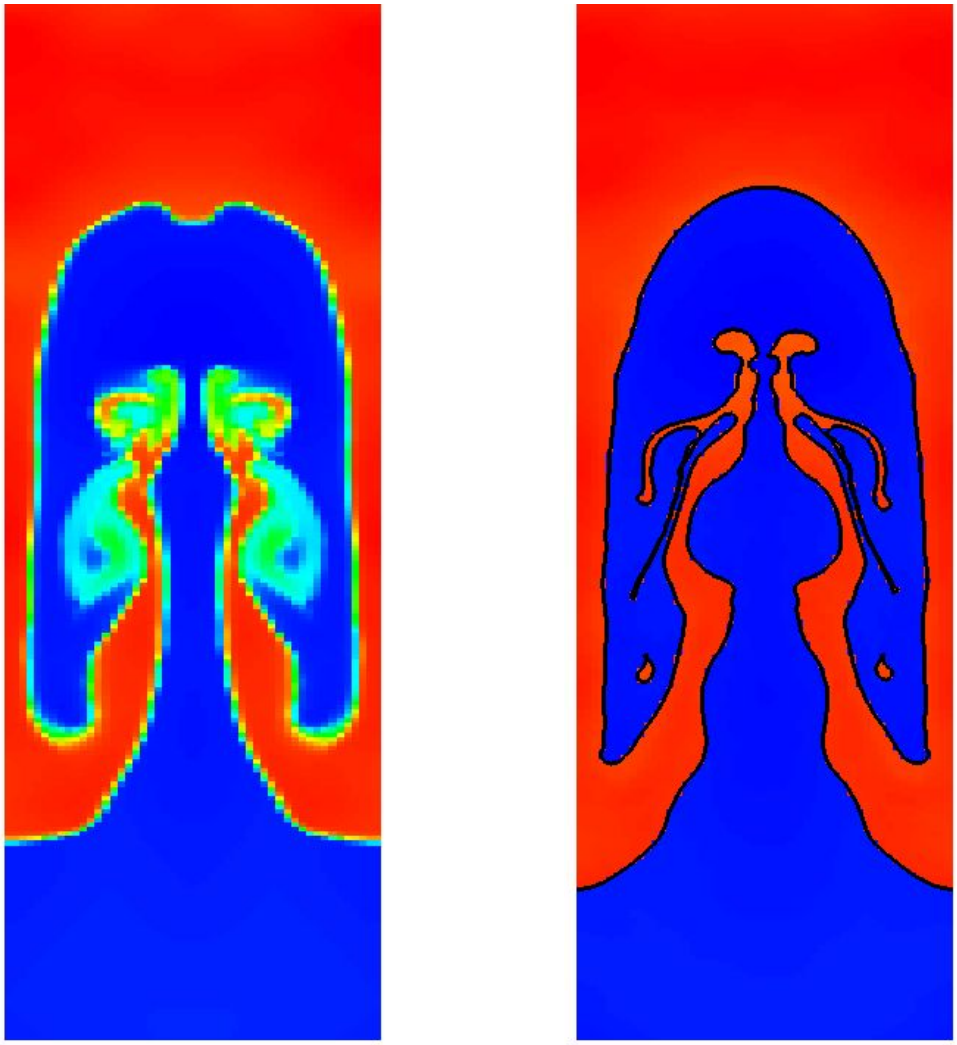

Figure 2: Rayleigh-Taylor single mode simulation at $t = 10$s. Comparison of FLASH run without front tracking (left frame) to a FLASH run with the use of the front tracking API (right frame). All diffusion occurring in the left frame is a result of numerical diffusion, prevented by the front tracking algorithm. The grid resolution is 50×250.

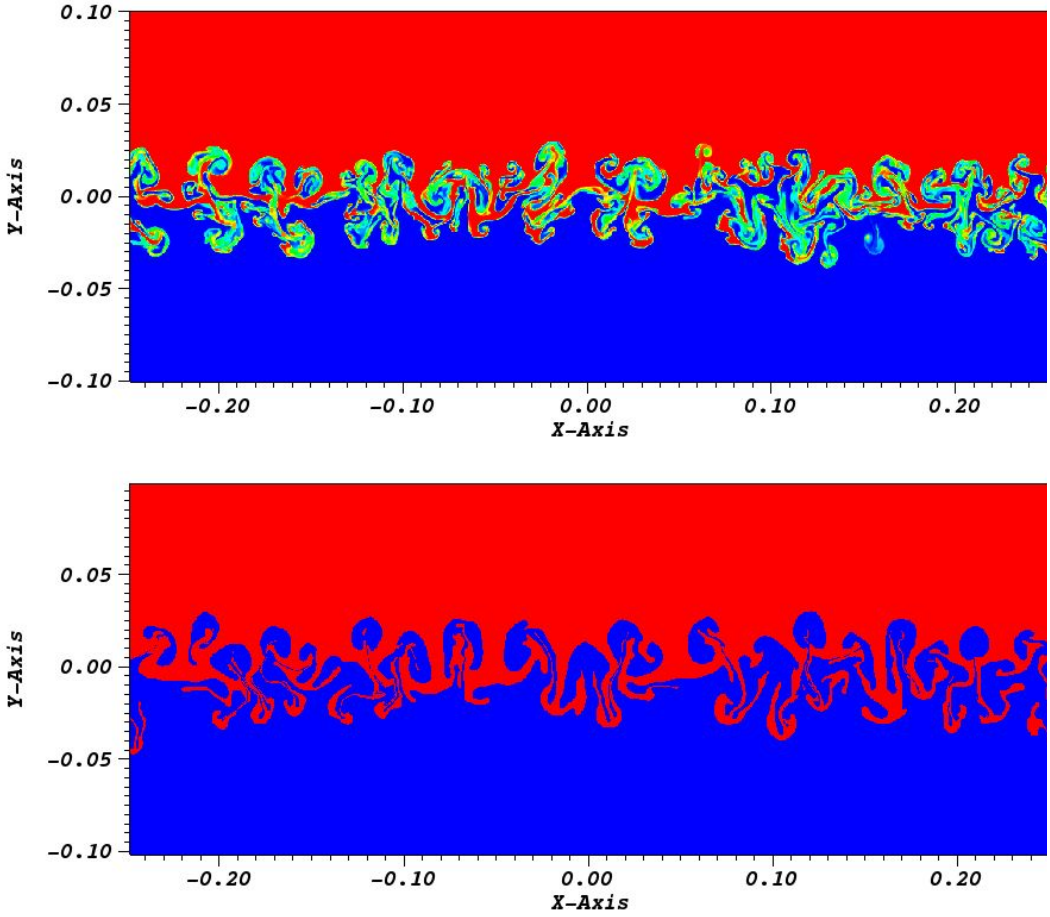

Figure 3: Rayleigh-Taylor multimode simulation at $t = 2.75$s. Comparison of FLASH run without front tracking (top frame) to a FLASH run with the use of the front tracking API (bottom frame). All diffusion occurring in the top frame is a result of numerical diffusion, prevented by the front tracking algorithm. The grid resolution is 1000×1000, providing ≈ 33 cells per smallest initial wavelength.

Bibliography

[1] W. Bo, X. Liu, J. Glimm, and X. Li, *A robust front tracking method: Verification and application to simulation of the primary breakup of a liquid jet*, SIAM J. Sci. Comput., 33 (2011), pp. 1505–1524.

[2] I.-L. Chern, J. Glimm, O. McBryan, B. Plohr, and S. Yaniv, *Front tracking for gas dynamics*, J. Comput. Phys., 62 (1986), pp. 83–110.

[3] P. Colella and H. M. Glaz, *Efficient solution algorithms for the riemann problem for real gases*, Journal of Computational Physics, 59 (1985), pp. 264 – 289.

[4] E. George and J. Glimm, *Self similarity of Rayleigh-Taylor mixing rates*, Phys. Fluids, 17 (2005), pp. 1–13. Stony Brook University Preprint SUNYSB-AMS-04-05.

[5] J. Glimm, D. H. Sharp, T. Kaman, and H. Lim, *New directions for Rayleigh-Taylor mixing*, Phil. Trans. R. Soc. A, 371 (2013), p. 20120183. Los Alamos National Laboratory Preprint LA-UR 11-00423 and Stony Brook University Preprint SUNYSB-AMS-11-01.

[6] R. Kaufman, H. Lim, and J. Glimm, *Conservative front tracking: the algorithm, the rationale and the API*, Bulletin of the Institute of Mathematics, Academia Sinica New Series, 11 (2016), pp. 115–130. Stony Brook University Preprint SUNYSB-AMS-15-01.

[7] H. Lim, J. Iwerks, J. Glimm, and D. H. Sharp, *Nonideal Rayleigh-Taylor mixing*, PNAS, 107(29) (2010), pp. 12786–12792. Stony Brook University Preprint SUNYSB-AMS-09-05 and Los Alamos National Laboratory Preprint LA-UR 09-06333.

[8] D. She, R. Kaufman, H. Lim, J. Melvin, A. Hsu, and J. Glimm, *Front tracking methods*, preprint submitted to Elsevier, (2016).

[9] G. Sod, *A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws*, J. Comput. Phys., 27 (1978), p. 1.