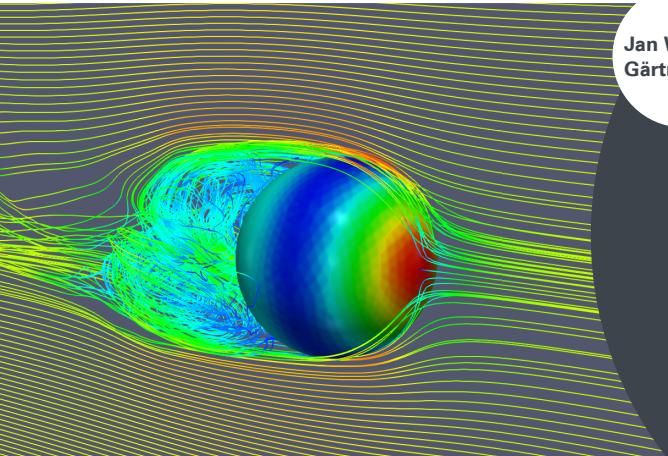
Universität Stuttgart

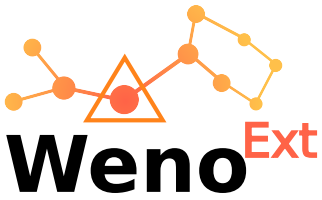


Jan Wilhelm
Gärtner

WENO Framework

Contributors

Universität Stuttgart



The **W**eighted **E**ssentially **N**on **O**scillating (WENO) scheme as implemented in this repository is originally written by,

Tobias Martin
tobimartin2@googlemail.com

Further main contributions are written by,

Jan Wilhelm Gärtner
jan.gaertner@outlook.de

The theory and implementation technique is based on several different authors mentioned within the attached Master Thesis. One main resource is the work of Pringuey [1].

When using this work please cite:

Publications:

J. W. Gärtner, A. Kronenburg, and T. Martin, "SoftwareX Efficient WENO library for OpenFOAM," SoftwareX, vol. 12, p. 100611, 2020, doi: 10.1016/j.softx.2020.100611.

Motivation

Simulating flows which contain discontinuities such as,

- shocks in supersonic flows,
- interfaces of two fluids

is a challenging task and requires numerical schemes that can give a high accuracy and yet, have enough dissipation around the discontinuity to avoid Gibbs oscillations.
To solve this conflicting requirements Weighted Essentially Non Oscillating (WENO) schemes have been developed that give a high accuracy within smooth regions and increase the dissipation around discontinuities.

Even though WENO schemes are widely used and have been established way over 20 years no WENO scheme implementation is available for OpenFOAM! Further, most implementations work on structured grids or only for finite difference methods. Here, we present a version that can handle any kind of mesh and can easily be included into any solver in OpenFOAM.

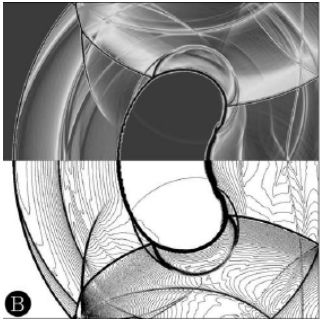


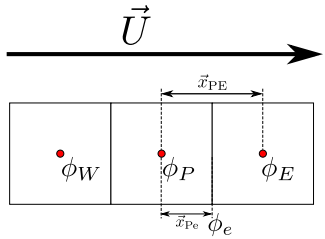
Figure: Shock wave interacting with helium bubble at Ma=1.22 using a WENO scheme [2].

Numerical Schemes Overview

In OpenFoam several numerical schemes are available for the surface interpolation of the cell centers onto the cell faces. Yet, schemes to resolve discontinuities, such as shocks, with a high order are not present. In the following important schemes of OpenFOAM are listed,

Scheme	Discretization	Accuracy
linear	$\phi_e = \frac{\vec{x}_{\mathsf{Pe}}\phi_P + \vec{x}_{\mathsf{eE}}\phi_E}{\vec{x}_{\mathsf{PE}}}$	2nd order
upwind	$\phi_e = \phi_P$	1st order
linearUpwind	$\phi_e = \phi_P + \vec{x}_{\mathsf{Pe}}\nabla\phi$	Up to 2nd order
WENO	$\phi_e = \frac{1}{\lvert V\rvert}\int_V \phi(\vec{x})\mathrm{d}x\mathrm{d}y\mathrm{d}z$	higher order

The WENO scheme can reach potentially 3rd order of accuracy or higher depending on the polynomial representation of $\phi(\vec{x})$.



WENO Reconstruction Method

3

Polynomial and Degree of Freedom

The WENO schemes replaces the cell centered value ϕ_i by a polynomial representation,

$$\phi_i = \frac{1}{|V_i|}\int_{V_i} \phi_i(\vec{x})\mathrm{d}x\mathrm{d}y\mathrm{d}z = \underbrace{\frac{1}{\left|V_i'\right|}\int_{V_i'} p_i(\vec{\xi})\mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\zeta}_{\text{reference space}}, \tag{1}$$

where the cell is transformed into a reference space $\vec{x} = \vec{x}(\xi, \eta, \zeta)$ to avoid scaling errors.
The polynomial $p_i(\xi)$ is represented by a set of basis functions,

$$p_i(\xi) = \phi_i + \sum_{k=1}^{K} a_k \Omega_k(\xi), \tag{2}$$

where K is the degree of freedom which depends on the order of the polynomial r,

$$K = \frac{(r+1)(r+2)(r+3)}{6} - 1. \tag{3}$$

This means that for a order of 3 already 19 coefficients a_k are needed!

Weighted Combination of Polynomials

To achieve a smooth solution a weighting of the polynomials is performed,

$$p_i(\xi) = \sum_{m=0}^{N} \omega_m p_m(\xi), \tag{4}$$

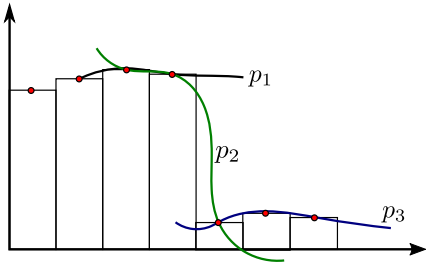
where N is the number of stencils and with the non linear weights,

$$\omega_m = \frac{\gamma_m}{\sum_{m=0}^{N} \gamma_m}. \tag{5}$$

Here γ_m is defined as,

$$\gamma_m = \frac{d_m}{\left(\epsilon + \mathcal{I}_{S,m}\right)^p}. \tag{6}$$

ϵ is a small value to avoid division by zero and $\mathcal{I}_{S,m}$ is the smoothness indicator function for the stencil m. d_m is used to increase the weight of the central stencil.



Polynomial Basis Functions

The WENO polynomial $p_{\text{WENO}}(\xi)$ can now be expressed as,

$$p_{\text{WENO}}(\xi) = \sum_{m=0}^{N} \omega_m \left(\phi_i + \sum^{K} a_k^{(m)} \Omega_k(\xi) \right). \tag{7}$$

Rearranging the equation leads to,

$$p_{\text{WENO}}(\xi) = \phi_i + \sum_{k=1}^{K} \left(\sum_{m=0}^{N} \omega_m a_k^{(m)} \right) \Omega_k(\xi) = \phi_i + \sum_{k=1}^{K} \tilde{a}_k \Omega_k(\xi) \tag{8}$$

The basis functions Ω_k are only dependent on the geometry and can be pre-calculated with,

$$\Omega_k(\xi) = \Psi_k(\xi) - \frac{1}{\left|V_i'\right|} \int_{V_i'} \Psi_k(\xi) \mathrm{d}\xi \mathrm{d}\eta \mathrm{d}\zeta. \tag{9}$$

The basis function $\Psi_k(\xi)$ is a result of a Taylor expansion around the center of V_i and is,

$$\Psi_k(\xi) = (\xi - \xi_i)^n (\eta - \eta_i)^m (\zeta - \zeta_i)^o. \tag{10}$$

Calculate the Coefficients a_k

Evaluating the WENO polynomial of Eq. (8) for an adjacent cell j gives,

$$\phi_j = \frac{1}{\left|V_j'\right|} \int_{V_j'} p_i(\xi) \mathrm{d}\xi \mathrm{d}\eta \mathrm{d}\zeta \tag{11}$$

$$\phi_j - \phi_i = \sum_{k=1}^{K} a_k^m \left(\frac{1}{\left|V_j'\right|} \int_{V_j'} \Omega_k(\xi) \mathrm{d}\xi \mathrm{d}\eta \mathrm{d}\zeta \right) \tag{12}$$

leading to an equation system of,

$$\phi_j - \phi_i = \sum_{k=1}^{K} \mathcal{A}_{jk} a_k^{(m)} \tag{13}$$

The matrix $\mathcal{A}$ only depends on known geometrical integrals and can be precomputed.
Storing these matrices are the cause of the huge memory demand of this scheme. For a third order WENO scheme already 19x38 entries have to be stored for each stencil and cell. Leading to a memory demand of 40GB for 1 million cells. If no reduction is possible, see implementation section!

WENO Upwind Scheme

The presented equations can now be easily included into the corrected upwind scheme, similar as the linearUpwind scheme of OpenFOAM. Using the definition of the flux at one face F,

$$\int_{F_l} \vec{n}_l \vec{u} \phi \mathrm{d}F_l = \underbrace{\vec{u}\vec{n}_l F_l}_{\text{vol. flux } Q} \frac{1}{F_l'} \int_{F_l'} \phi \mathrm{d}F_l' \tag{14}$$

$$= Q \left[\phi_i + \underbrace{\frac{1}{F_l'} \left(\sum_{k=1}^{K} \tilde{a}_k \int_{F_l'} \Omega_k(\xi) \mathrm{d}F_l' \right)}_{\text{explicit correction}} \right] \tag{15}$$

OpenFoam Implementation

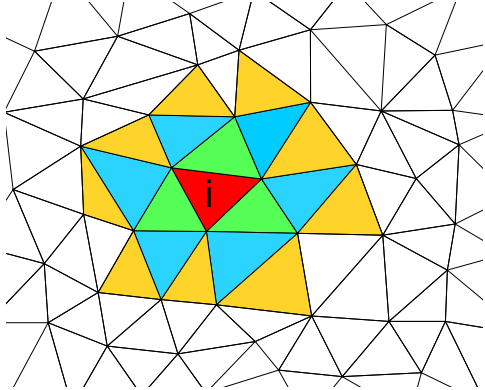
4

Creating Stencil List

The coefficients α_k are calculated on a compact set of stencils surrounding the current cell. The generation of this stencil list is done in three steps.

1) A big central stencil with n-layers of surrounding cells is generated.
2) Split the central stencil into several sectorial stencils. A sector typically corresponds to a face of the current cell. Thus, for a tetrahedral cell 4 faces equal 4 sectorial stencil plus the central stencil.
3) Cut the sectorial stencil list to the necessary size

The stencils are hereby stored as a list of lists of cell IDs.



Matrix Condition

The matrix system,

$$\mathcal{A}x = b,$$

accuracy depends on the condition of $\mathcal{A}$.

Find Best Conditioned Matrix

Add iteratively new cells to the matrix until the condition of $\mathcal{A}$ goes up again. Return the best conditioned matrix system with the upper limit of $2K$ cells, where K is the degree of freedom.

Performance increase:

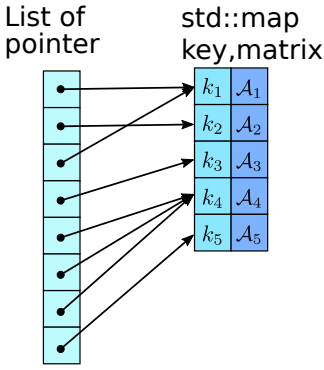
There is only a small performance increase. However, memory concerns are here the main driving point, see next slide.

Storing Inverse of $\mathcal{A}$

Problem:
Storing the inverse of the matrix system $\mathcal{A}^+$ for each stencil in each cell requires a lot of memory. In many cases meshes are not completely irregular and thus as $\mathcal{A}^+$ only depends on geometry information the memory demand can be reduced.

Matrix Data Bank System
To efficiently store the inverse matrices for all cells a new matrix data bank system is implemented. For each cell and each stencil only a pointer is stored, which references a matrix within the data bank implemented as a `std::multimap`.

Memory Reduction
For a 2D (400 cells) and 3D (64 000 cells) case using a regular grid of tetrahedrons, the memory demand can be substantially reduced:



Storing Inverse of $\mathcal{A}$

Memory Reduction

Memory reduction is tested on a cubic mesh with $100x100x100$ cells and one cylinder mesh (320k cells). Within the cylinder and the cubic mesh, mesh grading is applied.

Case	Memory Matrix Databank [GB]	Relative Reduction
no matrix DB	8	-
with matrix DB	0.1	98%
best conditioned	0.1	98%
grading & matrix DB	0.1	98%
Cylinder Mesh & no matrix DB	3.0	-%
Cylinder Mesh & matrix DB	0.2	93%

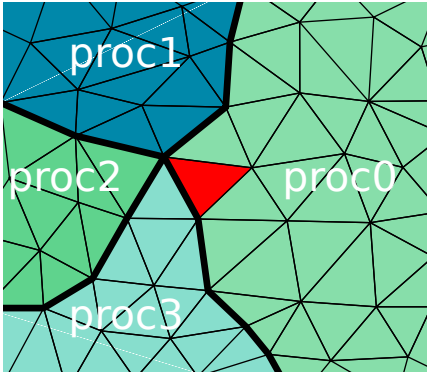
Parallelization in OpenFOAM

Parallelization of the WENO scheme is not as straight forward as for any other OpenFOAM scheme. In difference to other schemes it

- requires cell node values of more than the patch neighbor cell,
- requires information of processors that do not have a common boundary.

Example: The red marked cell in processor 4 requires the cell node values of processor 1, 2 and 3. Getting the cell nodal values raises following challenges:

- The light blue marked cells are not accessible through the patches as the extend further into the domain.
- Processor 1 does not share a boundary with processor 4 and thus cannot be found through the patch neighbor information of OpenFOAM



Parallelization in OpenFOAM

Pre-Processing:

The WENO scheme is solved for a decomposed domain with the following steps:

1) Each processor reconstructs a regional mesh of itself and neighboring processors.
   Implemented in class `globalFvMesh`
2) Collecting the stencils for the processor local cells on the global mesh and storing the global cell ID
3) Correcting the global cell ID with the local cell ID for cells within the own processor. Other cells are marked as non-local halo cells.
4) Map the correct processor ID to the halo cells
5) Store a list of cells required by other processors as well as the cells required by your own stencil list

Run-Time:

- At run time the cell nodal value of cells required by other processors are send and the halo cell data is received.
- Solve the matrix system of Eq. (13) to calculate the coefficients a_k
- Calculate the weighting factors ω and determine the weighted coefficients $\tilde{a}_k$.
- Calculate the explicit correction for the WENO upwind fit.

Reconstruct Regional Mesh

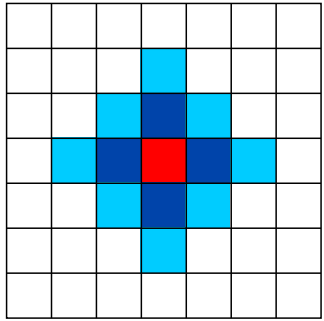
Find Neighbor processor

Every processor reconstructs the mesh of its own and neighboring processors. This is done in four steps:

1) Get the direct processor neighbors through the patch neighbor function, marked in dark blue.
2) Collect the neighbors of these processors, called second neighbors, marked in light blue.
3) The processor meshes are now read from their constant folder and combined to one regional mesh for each processor.

Memory Demand

The memory demand for reconstructing the regional mesh depend on the mesh size and decomposition. The memory demand differs also for 3D or 2D mesh.



Speed-Up with Vectorization

Loop vectorization:

Modern day processors and compilers can make use of vectorizing loops by applying the same operation to several data sets at once and thus increasing the speed of computation.

Vectorization and WENO Scheme:

During the run time of a simulation the calculation of the weights a_k is the most expensive operation. As this is only a simple matrix multiplication this can be well vectorized. However, OpenFOAMs data structure does not enable compiler vectorization! Therefore, a third party library `Blaze` is used.

Blaze Library:

Blaze is an open source, high-performance, C++ library for matrix manipulations. Therefore, using a blaze matrix object for storing the pseudo inverse matrices can make use of vectorization during the matrix multiplication.

Speed Test:

Calculating 200 times the divergence operator on a scalar field of 200x200 cells. The Blaze library gives a total speed up of 27%. (Tested on a AMD Ryzen 7 2700X.)

	Total Speed-Up	In Function
OpenFOAM	1.0	1.0
Blaze	1.27	1.4

Achieved Accuracy

5

Sinus Wave Test Case

Testing Divergence Operator

Calculate the divergence of,

$$\nabla \cdot (\boldsymbol{u}\phi),$$

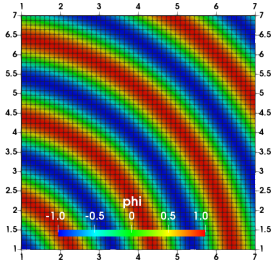
with,

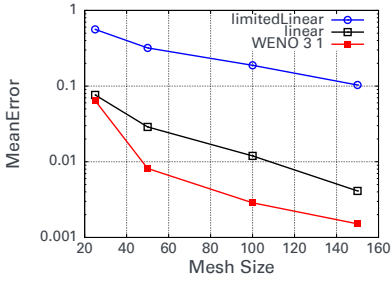
$$\boldsymbol{u} = \begin{bmatrix} \frac{x}{x^2+y^2} \\ \frac{y}{x^2+y^2} \end{bmatrix}, \qquad \phi = \sin(\sqrt{x^2+y^2}).$$

The error is calculated as the relative difference to the analytical solution.

Results

The results show that the WENO scheme with a third order polynomial performs, as expected, better than the linear scheme, which is a second order scheme.
This is only a more theoretical test case as the equations are solved explicitly and typically an implicit divergence operator is used.





Taylor Green Vortex Test Case

Case Setup

Accuracy within a simulation is tested with the Taylor Green Vortex test case ($\mathrm{Re} = 1600$) as defined by Wang et al. [3] in the 1st International Workshop on High-Order CFD Methods.
The reference solution is obtained by an dealiased pseudo-spectral code on a 512^3 grid.

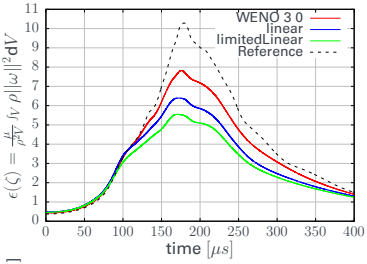
Mesh Study

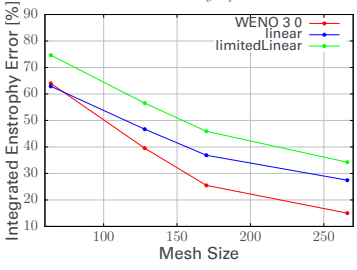
Simulations are run on an unstructured grid generated with the tool SALOME and the NetGen-3D algorithm.

Enstrophy

To compare the results the domain integrated enstrophy is compared to the reference case,

$$\epsilon(\zeta) = \frac{\mu}{\rho^2 V} \int_V \rho ||\omega||^2 \mathrm{d}V.$$





Universität Stuttgart

Jan Wilhelm Gärtner
Institut für Technische Verbrennung, Universität Stuttgart
Herdweg 51, 70174 Stuttgart, Germany

E-Mail: jan.gaertner@outlook.de
Phone: +49-711-685 674 69

References

[1] Thibault Roland Christophe Maurice Pringuey. *Large eddy simulation of primary liquid-sheet breakup*. PhD thesis, University of Cambridge, 2012.

[2] Eric Johnsen and Tim Colonius. Implementation of weno schemes in compressible multicomponent flow problems. *Journal of Computational Physics*, 219(2):715–732, 2006.

[3] Z.J. Wang, Krzysztof Fidkowski, Rémi Abgrall, Francesco Bassi, Doru Caraeni, Andrew Cary, Herman Deconinck, Ralf Hartmann, Koen Hillewaert, H.T. Huynh, Norbert Kroll, Georg May, Per-Olof Persson, Bram van Leer, and Miguel Visbal. High-order cfd methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72(8):811–845, 2013. doi: 10.1002/fld.3767.