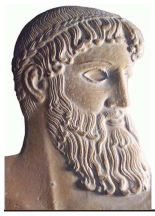

What is *ZEUS-3D*?

David A. Clarke
Professor of Astronomy and Physics
Institute for Computational Astrophysics
Saint Mary's University, Halifax NS, Canada
http://www.ica.smu.ca/zeus3d

May, 2016

Contents

1 Introduction

1.1 Overview

ZEUS-3D (version 3.6) is a multi-physics computational fluid dynamics (CFD) code written in *FORTRAN77* designed primarily for, but not restricted to, astrophysical applications. It was born from the *ZEUS*-development project headed by Michael Norman at the NCSA in the late 1980s and early 1990s whose principal developers included myself (1986–1988; `zeus04`, 1990–1992; *ZEUS-3D*), Jim Stone (1988–1990; *ZEUS-2D*), and Robert Fiedler (1992–1994; `ZEUSMP`). Both *ZEUS-2D* and *ZEUS-3D* (version 3.2; `zeus32`) were placed in the public domain in 1992, followed two years later by `ZEUSMP`, an *MPI* version of `zeus32`.

The NCSA version of the code was eventually absorbed by *ENZO* developed at UCSD, though this version of *ZEUS* itself underwent little development from `ZEUSMP`. Independently, this author has continued to develop *ZEUS* in its own right (Clarke, 1996; 2010), making it available to collaborators and others upon request since 1993. On completion of version 3.5 (when, after much angst, the MHD boundary conditions were finally stabilised), `dzeus35` (the "`d`" stands for "double precision") was made available on a publicly accessible website in 2007. This was the first re-release of the code by one of the original authors in more than a decade, and is a completely different code from `zeus32` and `ZEUSMP`, though still recognisable as a member of the *ZEUS* family of codes by its adherence to a staggered mesh and operator-splitting (both defined below). Development of the code and its algorithms continued, culminating it the release of version 3.6 (`dzeus36`) in 2016 on its own website.

1.2 Improvements over version 3.5

Version 3.6 includes a number of significant improvements and additions over version 3.5, including:

1. in-line 2-D plotting has been replaced with a "buffet-style" plotting, in which the user selects as many as two different scalars and three different vector fields to be plotted on the same image;
2. a non-cooling synchrotron aging algorithm (Nick MacDonald) has been introduced to allow "numerical observations" of the spectral index (via line-of-sight integrations performed by `RADIO`);
3. Lagrangian particles which carry a plethora of local variables as a function of time may now be inserted into the grid at $t = 0$ and/or during the simulation (Nick MacDonald, Mark Richardson);
4. slip-periodic boundary conditions have been introduced;
5. in-line movie making capability has been added (Stephen Campbell) for both pixel and `RADIO` dumps;
6. an FFT/FST self-gravity solver has been added (Logan Francis);
7. a one-fluid ambipolar diffusion module has been added (Chris MacMackin);
8. the transport step has been restructured—by updating primitive variables more frequently and resequencing the order in which some of the modules are called—to create a new *Finely Interleaved Transport* scheme. `FIT` solves a few long-standing numerical weaknesses in *ZEUS*, including "rarefaction shocks" (Falle, 2002) and potentially unstable 2-D advection and Alfvén wave transport. The old transport algorithm, now known as *Legacy Transport*, remains as an option in version 3.6.

Further, two new ancillary codes have been introduced (in addition to existing ancillary codes `PLOTZ` and `RADIO` which, respectively, allow post-run production of 1-D/2-D plots and line-of-sight integrations):

1. `ANIM8Z` allows post-run production of `mp4` movie files (where a palette and transfer function may be specified) using the new 4-byte movie tar-balls generated by `dzeus36` during a run.
2. `TESTZ` (Chris MacMackin) allows the code to be taken through its substantive test suite of more than 200 problems automatically, with differences between the current test and previous test reported. Thus, unlike version 3.5, this version comes with the entire *ZEUS* test-suite.

For more details on `PLOTZ`, `RADIO`, `ANIM8Z`, and `TESTZ`, see the *User Manual for ZEUS Ancillaries*.

The rest of this document gives an overview of some of the capabilities of the code, the algorithms used, and addresses some of the criticisms the code has received since its first release. This document is not a "how-to" manual for using *ZEUS*; for that, the reader is directed to the *ZEUS-3D User Manual*, which may also be found in the `documents` directory in the `dzeus36` tar-ball. Instead, this document introduces the user to the functionality of `dzeus36`, and addresses some of the technical issues of its design.

2 Physics

As a multi-physics and covariant CFD code, *ZEUS-3D* includes the effects of: magnetism; self gravity; Navier-Stokes viscosity; a second cospatial and diffusive fluid; molecular cooling; ambipolar diffusion; and isothermal, adiabatic, or polytropic equations of state. Terms in equations (1)–(6) below are colour-coded respectively; black terms being the standard equations of ideal HD. Simulations may be performed in any of Cartesian (`XYZ`), cylindrical (`ZRP`), or spherical polar coordinates (`RTP`), and in any dimensionality (1-D, $1\frac{1}{2}$-D, 2-D, $2\frac{1}{2}$-D, or 3-D), with half-dimensionality meaning the vector components in the symmetry direction(s) is (are) included. The set of equations solved are:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\,\vec{v}) = 0; \tag{1}$$

$$\frac{\partial\vec{s}}{\partial t} + \nabla\cdot\Big(\vec{s}\vec{v} + (p_1 + p_2 + p_B)\,\mathsf{I} - \vec{B}\vec{B} - \mu\,\mathsf{S}\Big) = -\rho\,\nabla\phi; \tag{2}$$

$$\frac{\partial e_1}{\partial t} + \nabla\cdot(e_1\vec{v}) = -p_1\nabla\cdot\vec{v} + \mu\mathsf{S}:\nabla\vec{v} - \mathcal{L} - \vec{v}\cdot\vec{L}; \tag{3}$$

$$\frac{\partial e_2}{\partial t} + \nabla\cdot(e_2\vec{v} - \mathsf{D}\cdot\nabla e_2) = -p_2\nabla\cdot\vec{v}; \tag{4}$$

$$\frac{\partial e_{\rm T}}{\partial t} + \nabla\cdot\Big[(e_{\rm T}+p_1+p_2-p_B)\,\vec{v} - \mu\mathsf{S}\cdot\vec{v} - \mathsf{D}\cdot\nabla e_2 + \vec{E}\times\vec{B} + \beta_{\rm AD}B^2\vec{L}\Big] = -\mathcal{L} + \beta_{\rm AD}L^2; \tag{5}$$

$$\frac{\partial\vec{B}}{\partial t} + \nabla\times(\vec{E} - \beta_{\rm AD}\vec{L}\times\vec{B}) = 0, \tag{6}$$

where:

ρ	is the matter density;
$\vec{v}$	is the velocity;
$\vec{s}$	is the momentum density $= \rho\vec{v}$;
p_1 & p_2	are the partial pressures from the first and second fluids;
p_B	is the magnetic pressure $= \frac{1}{2}B^2$;
I	is the unit tensor;
$\vec{B}$	is the magnetic induction (in units where $\mu_0 = 1$);
μ	is the coefficient of shear viscosity;
S	is the shear tensor whose elements, S_{ij}, are given by: $$S_{ij} = \partial_j v_i + \partial_i v_j - \tfrac{2}{3}\delta_{ij}\nabla\cdot\vec{v};$$ where ∂_i indicates partial differentiation with respect to the coordinates, x_i, $i = 1,2,3$, and where δ_{ij} is the usual “Kronecker delta”;
ϕ	is the gravitational potential, $\nabla^2\phi = 4\pi\rho$, in units where $G = 1$;
e_1 & e_2	are the internal energy densities of the first and second fluids;
$\mathcal{L}$	is the cooling function (of ρ and e_1) for nine coolants (HI, HII, CI, CII, CIII, OI, OII, OIII, SII) interpolated from tables given by Raga *et al.* (1997);
D	is the (diagonal) diffusion tensor;
$e_{\rm T}$	is the total energy density $= e_1 + e_2 + \frac{1}{2}\rho v^2 + \frac{1}{2}B^2 + \phi$;

$\vec{E}$	is the induced electric field $= -\vec{v} \times \vec{B}$ (note that $\vec{B} \times \vec{E}$ is the Poynting flux);
$\beta_{\rm AD}$	is the ambipolar diffusion coupling coefficient for ions and neutral particles;
$\vec{L}$	is the Lorentz force $= (\nabla \times \vec{B}) \times \vec{B}$;

and where the ideal equations of state close this system of equations:

$$p_1 = (\gamma_1 - 1)e_1; \qquad p_2 = (\gamma_2 - 1)e_2,$$

with γ_1 and γ_2 being the ratios of specific heats for the two fluids.

Radiation MHD using the flux-limited diffusion approximation (*e.g.*, Turner & Stone, 2001) is only partially implemented and thus not included above. The Raga cooling functions are available upon request in a separate *change deck*[1], but are not fully debugged. Note that one uses *either* the internal energy equation, (3), *or* the total energy equation, (5), depending upon whether a strictly positive-definite internal energy density, e_1, or strict conservation of energy, $e_{\rm T}$ (without cooling and ambipolar diffusion), is paramount. (See problem #22 in the 1-D *Gallery* for further discussion.)

The coefficient of shear viscosity, μ, and the shear tensor, S, can be structured to include the von Neumann-Richtmyer (vNR) artificial viscous terms used in *ZEUS* (von Neumann & Richtmyer, 1950; see also Stone & Norman, 1992a; Clarke, 1996; 2010). In particular,

$$\mu_{\rm vNR} = q_1 \delta x_i \rho c_{\rm s} - q_2 \delta x_i^2 \rho \min(0, \partial_i v_i) \qquad \text{and} \qquad \mathsf{S}_{\rm vNR} = \nabla \vec{v},$$

where q_1 and q_2 ($= \frac{1}{2}$`qlin` and $\frac{1}{2}$`qcon` in *ZEUS*) set respectively how much *linear* and *quadratic* artificial viscosity to apply, δx_i is the zone size in the i-direction, and $c_{\rm s}$ is the sound speed. Thus, artificial viscosity is applied in a momentum-conservative fashion, appearing as it does in a divergence term in equation (2).

3 Features and Algorithms

3.1 *ZEUS* as a conservative code

As all members of the *ZEUS* family of codes, `dzeus36` uses a *staggered mesh* and is *operator-split*. On a staggered mesh (look ahead to Fig. 2), primary scalars (ρ, e_1, e_2, $e_{\rm T}$) are zone-centred, components of primary vectors ($\vec{v}$, $\vec{B}$) are face-centred, and components of secondary vectors ($\vec{E}$) are edge-centred.

In an operator-split scheme, each term in equations (1)–(6) are accounted for sequentially, with the primitive variables updated after each term is accounted for (`FIT`) or after certain subsets of terms have been accounted for (Legacy transport). Operationally, the terms are divided into three sub-groups:

1. *Source terms* include all terms on the right hand side of equations (1)–(6), and are evaluated in the *source step* (routine `scrstep` in `dzeus36`);

2. Transport terms include all divergences and curls on the left hand side of equations (1)–(6). Of those, *compressive transport terms* (routines `ctran*`) include transport of all scalars (*e.g.*, $\partial_1 \rho v_1$) and vector components parallel to the transport direction (*e.g.*, $\partial_1 s_1 v_1$). Compressional terms are *unidirectional*.

3. *Transverse transport terms* (routines `ttran*`) include transport of all vector components perpendicular to the direction of transport (*e.g.*, $\partial_1 s_2 v_1$), all induction terms (*e.g.*, $\partial_2 E_3$), all transverse Lorentz force terms (*e.g.*, $B_1 \partial_1 B_2$), and the Poynting flux. Transverse transport terms are *bidirectional*.

[1] A change deck is a text file containing only the changes to the `FORTRAN` plus logic read by a precompiler indicating where the changes go in the code. See the *ZEUS-3D User Manual* for full operational details.

Operator splitting also includes *directional* and *planar splitting*. The source and compressional transport steps are directionally-split (sweeps of x_1, x_2, and x_3 done separately), whereas the transverse transport step is *planar-split* (interpolations and transport performed separately and implicitly on the x_1-x_2, x_2-x_3, x_3-x_1 planes). Most computational fluid dynamicists will be familiar with directional-splitting, but may not be familiar with planar-splitting. For that, the interested reader is referred to the document *What is Planar Splitting?*, and/or Clarke (1996) for details.

Other than the self-gravity term, equation (2) is written entirely in conservative form, with all terms on the LHS appearing as a perfect divergence. This includes the Lorentz force, since:

$$\nabla \cdot (p_B \mathsf{I} - \vec{B}\vec{B}) \;=\; \cancel{\nabla p_B} - (\nabla \times \vec{B}) \times \vec{B} - \cancel{\tfrac{1}{2}\nabla B^2} - \vec{B}\cancelto{0}{\nabla \cdot \vec{B}} \;=\; -\vec{L}, \tag{7}$$

using a well-known vector identity for $\nabla \cdot \vec{B}\vec{B}$. Note that this equality is predicated on $\nabla \cdot \vec{B} = 0$, and while *ZEUS* is designed so that a particular numerical differencing of $\nabla \cdot \vec{B}$ is always zero to within machine round-off error[2], this is not the numerical differencing of $\nabla \cdot \vec{B}$ that equation (7) requires for machine round-off accuracy. To wit, equation (7) requires that:

$$\nabla \cdot \vec{B}\vec{B} - (\nabla \times \vec{B}) \times \vec{B} - \tfrac{1}{2}\nabla B^2 \;=\; 0,$$

which, because of all the different centring of $\vec{B}$ one must do to evaluate each term, can only be true to within truncation error, not machine round-off error.

Alternately, using a vector identity that does not require $\nabla \cdot \vec{B} = 0$, we can write:

$$\vec{L} \;=\; (\nabla \times \vec{B}) \times \vec{B} \;=\; (\vec{B} \cdot \nabla)\vec{B} - \tfrac{1}{2}\nabla B^2,$$

which, in Cartesian coordinates, gives for the 1-component:

$$L_1 \;=\; \cancel{B_1 \partial_1 B_1} + B_2 \partial_2 B_1 + B_3 \partial_3 B_1 - \tfrac{1}{2}\partial_1\big(\cancel{B_1^2} + B_2^2 + B_3^2\big). \tag{8}$$

While the latter two surviving terms—being perfect derivatives—*are* in conservative form (for Cartesian coordinates, at least), the former two are not. In order to render these first two terms in conservative form, one must move B_2 and B_3 inside the derivatives which can be done by adding in $B_1 \nabla \cdot \vec{B}$:

$$\begin{aligned} L_1 \;&=\; B_2 \partial_2 B_1 + B_3 \partial_3 B_1 - \tfrac{1}{2}\partial_1\big(B_2^2 + B_3^2\big) + B_1 \partial_1 B_1 + B_1 \partial_2 B_2 + B_1 \partial_3 B_3 \\ &=\; \partial_2(B_2 B_1) + \partial_3(B_3 B_1) - \tfrac{1}{2}\partial_1\big(B_2^2 + B_3^2\big) + \tfrac{1}{2}\partial_1 B_1^2. \end{aligned} \tag{9}$$

Of course, once again, equations (8) and (9) will agree only to within truncation error.

Traditionally, *ZEUS* has used equation (8) to compute $\vec{L}$, rendering the momentum equation incompletely conservative. The first two (non-conservative) terms are accounted for in the transverse transport step, while the last two (conservative) terms are included in the source step. In `dzeus36`, both Legacy transport (`trnvrsn=0`) and `FIT` (`trnvrsn=1`) use equation (8), while an experimental conservative version of `FIT` (`trnvrsn=2`) uses equation (9). As such, `dzeus36` is the first version of *ZEUS* that can operate as a fully conservative code (to within machine-round-off error) by solving equations (2) and (5) as given. Much experimentation has been done to determine where exactly the new term, $\frac{1}{2}\partial_1 B_1^2$, must be sequenced in order for the code to still pass its test-suite (being unidirectional, it turns out to be in the compressive transport step, but not the source step), and while no test has yet revealed a problem with the conservative form of `FIT`, it still should be considered experimental and used with caution.

[2] There is no flexibility in how the divergence, curl, and gradient operators are differenced in *ZEUS*. These are determined by differencing Gauss' and Stokes' theorems in a manner that strictly enforces $\nabla \cdot \nabla \times \vec{A} = 0$ and $\nabla \times \nabla\phi = 0$ to within machine round-off error, as the conservative properties of the algorithm require.

3.2 *ZEUS* and Godunov-type codes

To be clear, *ZEUS-3D* is *not* a *Godunov*-type code, although it should also be made clear that a code need not be Godunov-type to be conservative. Based as it is on a staggered mesh, `dzeus36` computes fluxes for zone-centred quantities directly from the velocities at zone interfaces; no interpolations need be performed and no characteristic equations need be solved to obtain these velocities. In most cases, shocks and contacts can be captured in as few zones as in Godunov-type "shock-capturing" schemes (*e.g.*, Dai & Woodward, 1994; Ryu & Jones, 1995; Balsara, 1998; Falle *et al.*, 1998) in which all variables are zone-centred and fluxes are computed from the face-centred primitive variables determined by the (approximate) solution of the Riemann problem. See problem #22 in the 1-D *Gallery* for further discussion on this point.

Besides simplicity in coding, the principal advantage of the staggered mesh is the *solenoidal condition* on the magnetic field ($\nabla \cdot \vec{B} = 0$) is maintained trivially everywhere on a 3-D grid to within machine round-off error. Zone-centred Godunov-type schemes typically require a "flux-cleansing" step (*e.g.*, Dai & Woodward, 1994) or a diffusion step (*e.g.*, *FLASH*; Fryxell *et al.*, 2000) to eliminate or reduce numerically driven magnetic monopoles after each MHD cycle. "Hybrid" Godunov schemes with two sets of magnetic field components—one face-centred, one zone-centred—have also been developed (Balsara & Spicer, 1999; Gardiner & Stone, 2005) which require an additional round of two-point averaging and/or interpolation between the zone-centred and face-centred magnetic field components (or, more accurately, the fluxes and induced fields) in each MHD cycle. While flux-cleaning and hybrid methods both eliminate magnetic monopoles to machine round-off error, they introduce diffusion into the scheme whose effects, if any, have not been fully investigated. The diffusion step used in *FLASH* spreads monopoles around the grid rather than eliminating them, and this can pose problems in the vicinity of shocks where monopoles can build up faster than the diffusion step can remove them (Dursi, 2008, *private communication*).

ZEUS-3D is *upwinded* in the flow and Alfvén waves and *stabilised* (via the artificial viscosity) on compressional waves (sound waves in HD, fast and slow magnetosonic waves in MHD). Most Godunov-type methods also require some artificial viscosity to maintain stability though typically a factor of two or so less. As shock waves are self-steepening, shocks are typically captured by *ZEUS-3D* in two or three zones, although slow-moving shocks may typically be spread out over several more. Both contact discontinuities and Alfvén waves are diffused as they move across a grid, and will gradually spread over many zones. However, the code includes a contact steepener (Colella & Woodward, 1984) which, when engaged, introduces a local (to the discontinuity) "anti-diffusive" term that can maintain a contact within one or two zones. However and on occasion, the steepening algorithm can introduce small oscillations in the flow. To illustrate these points, problems #9–20 in the *ZEUS-3D* 1-D Gallery give the `dzeus36` solutions to the full suite of 1-D Riemann problems studied by Ryu & Jones (1995).

3.3 Critiques

Falle (2002) points out that *ZEUS* will, on occasion, attempt to fit a rarefaction wave with so-called "rarefaction shocks" (problem #21 in the 1-D gallery), and attributes this to the operator-split nature of the algorithm to solve the momentum equation (and specifically, because this makes the code first-order in time, but second order in space). Using Legacy transport—and thus for all versions of *ZEUS* previous to and including version 3.5—this could be ameliorated by applying sufficient artificial viscosity to correct the problem. However, as discussed on the web page for problem #21, `FIT` cures the problem of rarefaction shocks just about completely, without the need for additional artificial viscosity.

Falle (2002) also gives two examples of 1-D Riemann-type problems in which *ZEUS* converges to "incorrect values", and from this concludes "it is not satisfactory for adiabatic MHD". In fact, the failing is with how *ZEUS* was used, and not with the underlying MHD algorithm. Falle used *ZEUS* in its "non-conservative form" [*i.e.*, using the internal energy equation (3)] and this does indeed cause some variables, often the density and pressure, to converge to different levels than with a conservative code (see problem #22 in the 1-D gallery). In fact, since the analytic solution to the Riemann problem is normally obtained from the *conservative* equations, it shouldn't come as any surprise that the *non*-conservative equations arrive at different solutions in situations where the loss of energy is important. Indeed, *ZEUS-3D* in its non-conservative form arrives at *precisely the correct levels* to the non-conservative Riemann problem (with an appropriate model for energy loss), and comparisons with the conservative solutions are inappropriate. On the other hand, using the conservative total energy equation (5), as first suggested by Clarke (1996), `dzeus36` does, in fact, converge to the analytic solution of the conservative Riemann problem. Indeed and for the most part, these numerical solutions are just as "crisp" as fully upwinded schemes such as TVD (Ryu & Jones, 1995) or that developed and used by Falle (2002).

For a full response to the criticisms of the *ZEUS* algorithms levelled by Falle and others (*e.g.*, Tasker *et al.*, 2006), see Clarke (2010). See also the pages for problems #9–22 in the *ZEUS-3D* 1-D Gallery.

3.4 Geometry

The code is written in a "covariant" fashion, meaning key vector identities (*e.g.*, $\nabla \cdot \nabla \times \vec{A} = 0$; $\nabla \times \nabla \phi = 0$) are preserved to machine round-off error for the three most commonly used orthogonal coordinate systems: Cartesian (`XYZ`); cylindrical (`ZRP`); and spherical polar (`RTP`). The user should be aware that the conservative nature of the momentum equation (2) is true only in Cartesian coordinates where the gradient and divergence derivatives are identical. In curvilinear coordinates where the metric factors render the gradient and divergence derivatives distinct, so-called "pseudo-forces" in the momentum equation (and analogous magnetic terms from $\vec{L}$) introduce imperfect derivatives which render the algorithm non-conservative. This may have effects on some specialised and pedagogical 1-D and 2-D test problems but, in practise, should have little if any measurable consequence on a fully dynamic simulation.

In addition, the code is designed to run as efficiently in 1-D or 2-D as a specifically-written 1-D or 2-D code. That is to say, if the user wishes to run the code in its 2-D mode, one doesn't have to worry that the actual execution would be a 3-D code with a few zones in the symmetry direction. In fact, in its 2-D mode, *ZEUS-3D* will only compute the fluxes in the two active directions, and arrays will be declared completely as 2-D entities.

3.5 Induction algorithms

Four algorithms for solving the induction equation are included in this release. The original Evans and Hawley (1988) *Constrained transport* scheme (`CT`) was among the first such algorithms developed that could maintain the solenoidal condition to machine round-off error in 3-D, and was incorporated in *ZEUS-2D* in 1990. The Stone & Norman *Method of Characteristics* (`MoC`; Stone & Norman, 1992b) algorithm was built on top of `CT` in 1992 to allow inductive steps to be upwinded in the Alfvén speed, a critical attribute for any stable MHD code. Figures 1*a*) and 1*b*) show respectively the results after a 1-D Alfvén wave has propagated for 1.5 pulse-widths in either direction using `CT` without `MoC` and with `MoC`.

Shortcomings with `MoC` become apparent in multi-dimensions. Clarke (1996) showed that `MoC` is subject to "magnetic explosions", a numerical instability most evident in super-Alfvénic turbulence in which local kinetic and magnetic energy densities can interchange, thereby suddenly and catastrophically introducing a dynamically important magnetic field into an otherwise high-β flow inside a single MHD cycle. Clarke (1996) also showed that while this is a local effect, it can have serious global consequences. Integrated across the entire grid, `MoC` will cause the magnetic energy density to grow exponentially with e-folding times much shorter than can be explained physically by turbulence (*e.g.*, Clarke, 2010).

Hawley & Stone (1995) devised the so-called `HSMoC` in which the results from `CT` with `MoC` are simply averaged with those of `CT` without. Figure 1*c*) shows that an Alfvén wave transported by `HSMoC` is stable but, as implemented in this release, requires twice as many time steps as `MoC` alone to maintain that

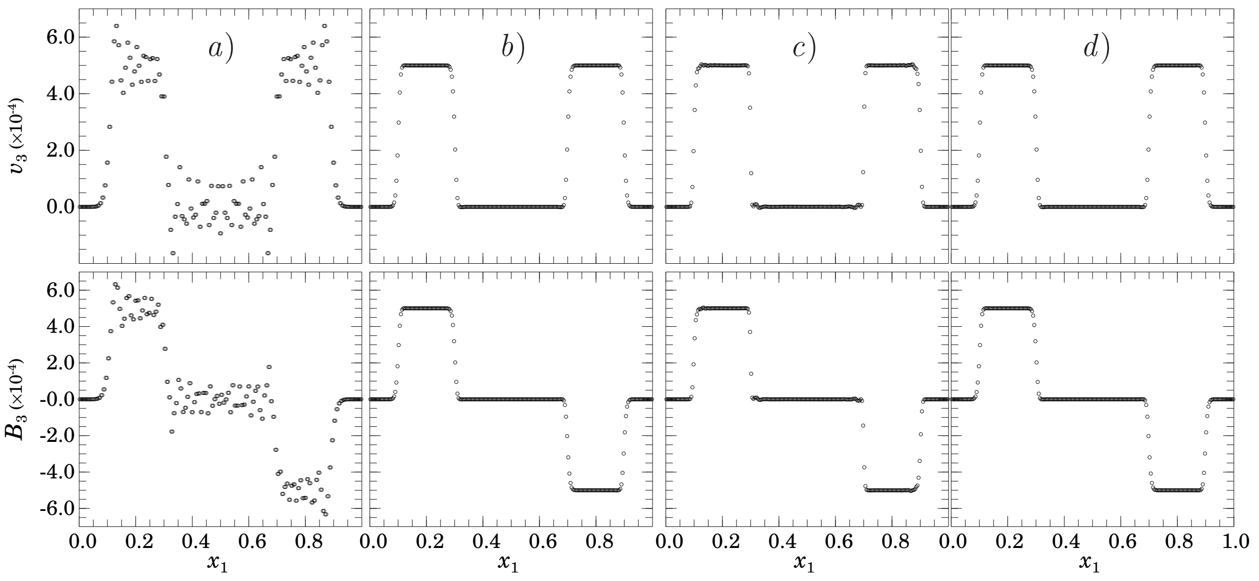

Figure 1: A square wave perturbation (10^{-3}) is applied to v_3 in the central 50 zones of a 1-D system resolved with 250 zones where $\rho = 1$, $p = 0.6$, $v_1 = 0$, and $B_1 = 1$, and allowed to propagate to $t = 0.75$. From left to right are the results for *a*) no upwinding in the Alfvén characteristics (original `CT` scheme); *b*) `MoC`; *c*) `HSMoC`; and *d*) `CMoC`, all described in the text.

stability (the "Courant number" has to be 0.5 or less). Still, Hawley & Stone (1995) claim `HSMoC` is more resilient to the "explosive" field instability than straight `MoC`, although it is unclear why this should be given that the hybrid scheme does not address the fundamental cause of the instability (described below).

The *Consistent Method of Characteristics* (`CMoC`; Clarke 1996) introduces a paradigm shift in how the induction equation is treated numerically. The basic principle of `MoC` remains intact: transverse interpolations of $\vec{v}$ and $\vec{B}$ are upwinded in the Alfvén characteristics. However, rather than doing this in a *directionally-split* fashion, `CMoC` is *planar-split* and this completely cures the magnetic explosion instability in super-Alfvénic turbulence.

The issue is this. In order to estimate the 3-component of the induced electric field ($E_3 = v_1B_2 - v_2B_1$ located at the 3-edge), upwinded 1-interpolations of v_2 and B_2 and upwinded 2-interpolations of v_1 and B_1—all to the 3-edge—are needed simultaneously (see Fig. 2). However, before the 1-interpolated 2-components can be computed, 2-interpolated 1-components are needed to construct the 1-characteristics while, at the same time, before the 2-interpolated 1-components can be computed, 1-interpolated 2-components are needed to construct the 2-characteristics. To break this apparent "catch-22", the directionally-split `MoC` makes preliminary estimates of all vector components at the 3-edges by taking two-point averages and using these to construct the required characteristics. The problem is, the two-point averages and subsequent upwinded values represent two independent estimates of the same quantities at the same location and time and these can be radically different particularly at shear layers. These differences are at the root of the explosive magnetic field instability in `MoC` and have to be eliminated to cure the problem.

To this end, the planar-split `CMoC` considers the entire 1-2 plane as one entity and identifies the upwinded *quadrant* relative to the 3-edge (rather than the upwinded *direction* as in the directionally-split `MoC`) thereby finding the 1- and 2-interpolations simultaneously. The rather elaborate algorithm to perform these implicit interpolations is described in detail in Clarke (1996), and rendered in *FORTRAN* in the `CMOC*` routines in `dzeus36`. The reader is also referred to the on-line document *What is Planar Splitting?* for a quick synopsis. As a result of planar-splitting, the 1-components used to construct the 1-characteristics for the

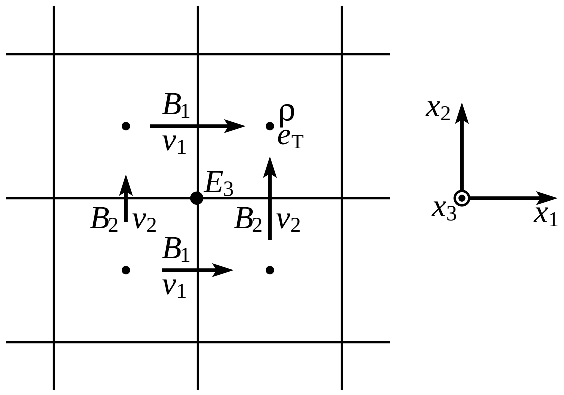

Figure 2: A segment of a Cartesian grid centred over the 3-edge where the 3-component of the induced electric field (E_3) is located. Shown also are two locations each for the 1- and 2-components of the velocity and magnetic field which must be interpolated to the 3-edge in order to make an estimate of E_3. In this projection, v_3 and B_3 (not shown) would appear co-spatial with the zone-centred quantities, ρ and e_T. In fact, they are staggered back and forward half a zone in the 3-direction.

1-interpolated 2-components are *consistent* with (the same as) the 2-interpolated 1-components, and *vice versa*, whence the leading ‘C’ in CMoC. Figure 1*d*) shows the 1-D Alfvén wave transported by CMoC and, as CMoC reduces to MoC in 1-D, it is no surprise that this plot is identical to Fig. 1*b*).

The original MoC as described by Stone & Norman (1992b)—embodied in the NCSA releases of zeus32 and ZEUSMP—consists of a rather complicated hybrid of a *Lagrangian step* for the transverse Lorentz forces followed by an *Eulerian* step for the induced fields and then a separate transverse momentum update whose fluxes are constructed from values upwinded in the flow speeds only. In fact, a much simpler scheme is possible involving a single Eulerian step whose interpolated values are used for all three purposes (transverse Lorentz forces, induced electric field, *and* transverse momentum transport), and this is how each of MoC, HSMoC, and CMoC are implemented in both dzeus35 and dzeus36.

Finally, the planar-split CMoC has been criticised by some as being *overly* complex, and indeed it is much more complicated to program than the MoC, for example. Still, the complexity of planar splitting over directional splitting is no more than the complexity of the curl (the operator in the induction equation) over the divergence (the operator in the hydrodynamical equations where directional splitting is appropriate), and thus planar splitting is no more complex than it needs to be. Of note is the fact that CMoC in dzeus36 has been used for more than two decades for a wide number of applications with no indication of numerical instability or “sensitivity” whatsoever in all dimensionalities and geometries.

3.6 Boundary conditions

A variety of MHD boundary conditions are supported. Boundaries are set independently for each of the six (in 3-D) boundaries and, within each boundary, the boundary *type* may be set zone by zone, if needed. The ten boundary types currently available are:

1. reflecting boundary conditions at a symmetry axis ($r = 0$ in ZRP, $\theta = 0$ or π in RTP) or grid singularity ($r = 0$ in RTP);
2. *non-conducting* reflecting boundary conditions, Fig. 3*a*);

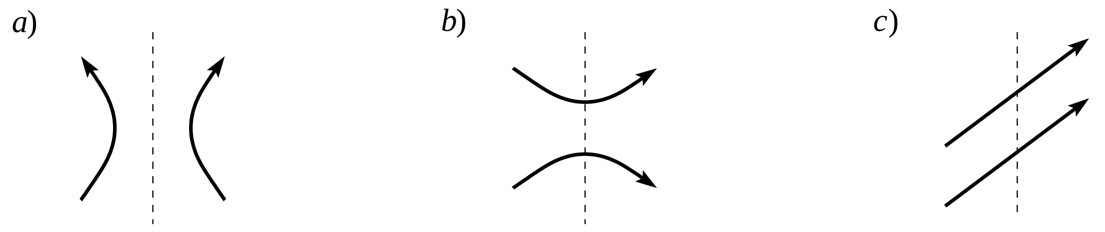

Figure 3: Panels show magnetic field lines across three different types of reflecting boundaries: a) non-conducting reflecting boundary conditions impose $B_{\|}(\text{out}) = -B_{\|}(\text{in})$, $B_{\perp}(\text{out}) = B_{\perp}(\text{in})$; b) conducting reflecting boundary conditions impose $B_{\|}(\text{out}) = B_{\|}(\text{in})$, $B_{\perp}(\text{out}) = -B_{\perp}(\text{in})$; and c) continuous reflecting boundary conditions impose $B_{\|}(\text{out}) = B_{\|}(\text{in})$, $B_{\perp}(\text{out}) = B_{\perp}(\text{in})$. The designations $\|$ and $\perp$ are relative to the boundary *normal*.

3. *conducting* reflecting boundary conditions, Fig. 3b);
4. *continuous* reflecting boundary conditions, Fig. 3c);
5. periodic boundary conditions;
6. "self-computing" boundary conditions (for *AMR*);
7. transparent (characteristic-based) outflow boundary conditions (not implemented);
8. "selective" inflow boundary conditions (useful for sub-fast-magnetosonic inflow);
9. traditional outflow boundary conditions (variables constant across boundary);
10. traditional inflow boundary conditions (suitable for super-fast-magnetosonic inflow).

All boundary conditions strictly adhere to the magnetic solenoidal condition, even where different boundary types may adjoin on the same boundary. The first five conditions may be considered *exact*, while the latter five are approximate and can launch undesired waves into the grid under some circumstances.

3.7 Self-gravity algorithms

Four algorithms for solving Poisson's equation are included in `dzeus36`: *Successive Over-Relaxation* (`SOR`), *Full Multi-Grid* (`FMG`), a fast-Fourier transform scheme (`FFT`), and a fast-sine transform scheme (`FST`). The former two algorithms existed in `dzeus35` and were taken from Press *et al.* (1992). `SOR` is an "N^2" algorithm and, while simple to program and understand, will dominate the computing time required for grids larger than 32^3 (*i.e.*, take longer than the MHD update). `FMG` is an example of an "N-log N" algorithm and, while rather complex, requires a comparable amount of computing time to the MHD cycle regardless of grid size. There is no measurable difference between the "accuracies" of `SOR` and `FMG`, and the only "nuisance" about `FMG` is that it requires the number of zones in each direction to be one less than a power of two. Neither scheme is compatible with periodic boundary conditions.

New to `dzeus36` are the fast-Fourier and fast-sine algorithms. Both are "N-log N" algorithms comparable in speed to `FMG`. The `FFT` scheme requires each direction be divided into a number of zones equal to a power of two, and implicitly *assumes* periodic boundary conditions. The `FST` scheme requires each dimension be divided into one zone less than a power of two (like `FMG`) and works with preset (not periodic) boundary values.

Boundary conditions for the gravitational potential are set independently from the MHD conditions described in §3.5. In addition to periodic boundary conditions that are implicit to the `FFT` scheme, two methods for specifying gravitational boundary conditions for the other gravity solvers are supported. Analytically determined boundaries may be set, or boundaries may be set by performing a multipole expansion (to include the *dotridecapole*) to the boundary. The latter requires that the vast majority of matter be contained well inside the bounded region.

3.8 Interpolation schemes

Numerous interpolation schemes (for determining fluxes) are available in `dzeus36`. Scalars may be interpolated with a first-order donor cell, one of three second order "van Leer" schemes (van Leer, 1977) including a "velocity corrected" scheme which takes into account the fact that the face-centred velocity can vary across the zone, and Colella & Woodward's (1984) third order *Piecewise Parabolic Interpolation* (`PPI`) scheme including their contact steepener. In *ZEUS-3D*, `PPI` has been modified with a *global extremum detector* which prevents global extrema from being monotonised (clipped) making it better suited for regions of smooth flow. This feature does not interfere with *PPI*'s renowned ability to maintain numerical monotonicity and to detect and steepen discontinuities. Vector components may be interpolated in the longitudinal direction using donor cell, van Leer interpolation, or `PPI`, while transverse interpolations (using the implicit planar-split method described in §3.5) may be donor cell or van Leer only.

3.9 Moving grid

`dzeus36` has inherited the "moving-grid" infrastructure designed by Jim Wilson and Michael Norman in the 1970s. It is a pseudo-Lagrangian technique in which a grid line can be given the average speed of flow across it, thereby reducing grid diffusion and increasing the time step. It is also useful in simulations such as a blast front, where the structures may be followed for great lengths of time without having to define an enormously long grid through which the front moves. While an attempt has been made to carry this feature with all the new developments over the past two decades, it has never been used or tested by this author. Therefore, any user wishing to use this feature is cautioned that some significant debugging and code development may be needed to restore it to its full capability. If anyone needing this utility elects to undertake this task, their change deck implementing their modifications into the code would be greatly appreciated by the developer so that it may be included—with due credit—in future releases of the code.

3.10 Graphics and I/O

Numerous I/O options are supported *in-line* by `dzeus36`. These include 1-D and 2-D graphics (*NCAR* and *PSPLOT*), full-accuracy dumps for check-pointing, time-slice files (ascii and graphics) to monitor variable extrema and certain integrated quantities over the course of the simulation, and *HDF* dumps (version 4.2), required for generating post-run graphics with ancillary programs `PLOTZ` and `RADIO`. *HDF* files can also be used by *IDL* and other commercially available visualisation packages.

Replacing the old-style pixel and `RADIO` dumps are movie features that produce both an `mp4` file ready for viewing, and a tar-ball created from sequences of 4-byte "pixel" and "`RADIO`" dumps, suitable for generating a new `mp4` with different colour palette and scaling (ancillary program `ANIM8Z`). Previously, one had to re-run the entire simulation to rescale the animations. Pixel dumps are 4-byte deep 2-D raster slices of various variables through the datacube, while `RADIO` dumps are line-of-sight integrations of various flow variables and "numerical observations" (*e.g.*, synchrotron emission) through the datacube.

Also new to `dzeus36` are Lagrangian particles, or "corks", which can be launched at any time anywhere on the grid and carry with them a host of measurements at numerous point of their trajectory. These quantities can be plotted as a function of time or position for further insight into the simulation.

3.11 Parallelism, efficiency, and memory occupancy

`dzeus36` is parallelisable for an SMP (shared memory platform) environment and the necessary *OpenMP* commands (and *FPP* commands under Cray's `UNICOS`) are automatically inserted into all routines—including those introduced by the user—by the precompiler *EDITOR* which comes with the download. The code is *not* written with *MPI* commands and thus is not suitable for a DMP (distributed memory platform) such as a "Beowulf" cluster. Parallel speed-up factors depend a great deal on the platform and how the grids are declared. On a 16-way SMP using a 120^3 grid, the code is about 98% parallel giving a speed-up factor of about 12.5.

Not including I/O, `dzeus36` uses about 1,500, 2,200, and 3,000 floating point operations ("flops") per MHD zone update (no self-gravity) for 1-D, 2-D and 3-D simulations respectively. These include all adds, multiplies, divides, if tests, subroutine calls, *etc.* required to update a single zone. For pure hydrodynamics, expect about half these number of flops; for self-gravitating MHD (using `FMG` or `FFT`), double. Thus, for example, a 256^3 3-D MHD simulation evolved for 25,000 time steps requires about $3 \times 10^3 \times 1.7 \times 10^7 \times 2.5 \times 10^4 \sim 1.3 \times 10^{15}$ flops to complete. On a single 2 GHz core with 50% efficiency, this translates to $\sim 1.3 \times 10^6$ s ($\sim$ 330,000 zone-cycles per second), or about 350 cpu hours. On a dedicated 16-way SMP with a speed-up factor of 12.5, the same simulation takes about 28 wall-clock hours.

Finally, the memory occupancy for a 3-D MHD (HD) calculation is equivalent to about 20 (16) 3-D arrays. For a 256^3 calculation, this means about $20 \times 1.7 \times 10^7 \times 8$ bytes (for double precision) $\sim$ 3 GBytes.

References

Balsara, D. S., 1998, ApJS, 116, 133.
Balsara, D. S. & Spicer, D. S., 1999, J. Comput. Phys., 149, 270.
Clarke, D. A., 1996, ApJ, 457, 291.
——., 2010, ApJS, 187, 119.
Colella, P., & Woodward, P. R., 1984, J. Comput. Phys., 54, 174.
Dai, W., & Woodward, P. R., 1994, J. Comput. Phys., 111, 354.
Evans, C. R., Hawley, J. F., 1988, ApJ, 332, 659.
Falle, S. A. E. G., 2002, ApJ, 557, L123.
Falle, S. A. E. G., Komissarov, S. S., & Joarder, P., 1998, MNRAS, 297, 265.
Fryxell, B., Olson, K., Ricker, P., Timmes, F. X., Zingale, M., Lamb, D. Q., MacNiece, P., Rosner, R., Truran, J. W., & Tofu, H., 2000, ApJS, 131, 273.
Gardiner, T. A. & Stone, J. M., 2005, J. Comput. Phys., 205, 509.
Hawley, J. F., & Stone, J. M., 1995, Comp. Phys. Comm., 89, 127.
Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P., 1992, *Numerical Recipes*, (Cambridge: Cambridge University Press).
Raga, A. C., Mellema, G., & Lundqvist, P., 1997, ApJS, 109, 517.
Ryu, D. & Jones, T. W., 1995, ApJ, 442, 228.
Stone, J. M., & Norman, M. L., 1992a, ApJS, 80, 759.
——., 1992b, ApJS, 80, 791.
Tasker *et al.*, 2008, MNRAS, 390, 1265.
Turner, N. J. & Stone, J. M., 2001, ApJS, 135, 95.
van Leer, B., 1977, J. Comput. Phys., 23, 276.
von Neumann, J., & Richtmyer, R. D., 1950, J. Appl. Phys., 21, 232.