LA-UR-20-22583
Approved for public release; distribution is unlimited.

Title: Non-Cartesian Coordinate Systems in the Portage Library

Author(s): Krueger, Brendan K.

Intended for: Include with software documentation for Portage library

Issued: 2020-03-27

Los Alamos NATIONAL LABORATORY EST. 1943

memorandum

XCP-2: Eulerian Codes

To: Angela Herring, XCP-1, MS T085
From: Brendan Krueger, XCP-2, MS T087
Phone: (505) 667-0243
Symbol: XCP-2:20-001
Date: February 24, 2020

Subject: Non-Cartesian Coordinate Systems in the Portage Library

In order to be more broadly useful, the Portage framework needs to be able to support non-Cartesian coordinate systems. This document will lay out the most important details of implementing non-Cartesian coordinate systems in Portage.

The most immediate need for this feature is integration with the EAP code base. Towards that end, the focus will be on certain curvilinear coordinate systems. However, this will provide a framework that can be used to implement other coordinate systems[1].

I. Curvilinear Coordinate System Conventions

There are multiple coordinate system conventions for curvilinear coordinates, and in practice most people think of their convention as the "One True Convention". The closest thing to a standard appears to be ISO 80000-2:2009. As any choice of convention will bother many readers, we will follow this standard[2]. The standard is a right-handed coordinate system with variables as shown in Table (1). It is assumed that all rotational coordinates are in radians. If you choose to use another measure for angle (e.g., degrees), then you may have to rederive some quantities. A visualization of this convention is shown in Figure (1).

variables	
symbol	meaning
x	the first Cartesian coordinate
y	the second Cartesian coordinate
z	the third Cartesian coordinate; $\hat{\boldsymbol{x}} \times \hat{\boldsymbol{y}} = \hat{\boldsymbol{z}}$
r	the distance from the origin (spherical radius)
ρ	the distance from the z axis (cylindrical radius)
ϕ	the azimuthal angle (around the z axis, starting at the positive x axis)
θ	the angle of declination (descending from the positive z axis)

Table 1: Coordinate system variables as defined by ISO 80000-2:2009.

[1] Properties of general orthogonal curvilinear coordinate systems, along with a handful of examples beyond those discussed here, are summarized nicely at `https://en.wikipedia.org/wiki/Orthogonal_coordinates`.

[2] This choice seems to bother nearly all of the Portage team, but the preferred convention varies, reinforcing the idea that there is no widely-accepted standard. Making everyone equally angry seems a reasonable compromise.

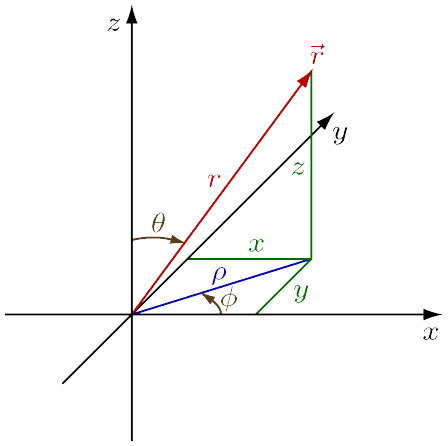

Figure 1: Coordinate system conventions used in this document, based on standards defined in ISO 80000-2:2009.[3]

II. Coordinate Systems

The coordinate systems to be implemented are Cartesian, cylindrical, and spherical. The relationships between coordinate variables are shown in Table (2), and unit vectors are shown in Table (3). Notice that while $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, and $\hat{\boldsymbol{z}}$ are constant, the other unit vectors are position dependent.

		from		
		Cartesian	cylindrical	spherical
to	Cartesian		$x = \rho\cos(\phi)$ $y = \rho\sin(\phi)$ $z = z$	$x = r\sin(\theta)\cos(\phi)$ $y = r\sin(\theta)\sin(\phi)$ $z = r\cos(\theta)$
	cylindrical	$\rho = \sqrt{x^2 + y^2}$ $\phi = \arctan\left(\frac{y}{x}\right)$ $z = z$		$\rho = r\sin(\theta)$ $\phi = \phi$ $z = r\cos(\theta)$
	spherical	$r = \sqrt{x^2 + y^2 + z^2}$ $\theta = \arctan\left(\frac{\sqrt{x^2 + y^2}}{x}\right)$ $\phi = \arctan\left(\frac{y}{x}\right)$	$r = \sqrt{\rho^2 + z^2}$ $\theta = \arctan\left(\frac{\rho}{z}\right)$ $\phi = \phi$	

Table 2: Curvilinear coordinate system conversions.

[3] This is a public domain image from Wikimedia Commons and can be found at `https://commons.wikimedia.org/wiki/File:Physics_Coordinates.png`.

unit vectors	
cylindrical	spherical
$\hat{\boldsymbol{\rho}} = \cos(\phi)\,\hat{\boldsymbol{x}} + \sin(\phi)\,\hat{\boldsymbol{y}}$	$\hat{\boldsymbol{r}} = \sin(\theta)\cos(\phi)\,\hat{\boldsymbol{x}} + \sin(\theta)\sin(\phi)\,\hat{\boldsymbol{y}} + \cos(\theta)\,\hat{\boldsymbol{z}}$
$\hat{\boldsymbol{\phi}} = -\sin(\phi)\,\hat{\boldsymbol{x}} + \cos(\phi)\,\hat{\boldsymbol{y}}$	$\hat{\boldsymbol{\theta}} = \cos(\theta)\cos(\phi)\,\hat{\boldsymbol{x}} + \cos(\theta)\sin(\phi)\,\hat{\boldsymbol{y}} - \sin(\theta)\,\hat{\boldsymbol{z}}$
$\hat{\boldsymbol{z}} = \hat{\boldsymbol{z}}$	$\hat{\boldsymbol{\phi}} = -\sin(\phi)\,\hat{\boldsymbol{x}} + \cos(\phi)\,\hat{\boldsymbol{y}}$

Table 3: Curvilinear unit vectors in terms of Cartesian unit vectors. The $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, and $\hat{\boldsymbol{z}}$ unit vectors are constant, but the others depend on position.

II.A. Cartesian Coordinates

Cartesian coordinates are the simplest case, and Portage is already implemented assuming Cartesian coordinates. We may not need the full detail for Cartesian coordinates, but it provides a nice comparison point to see how the more complex coordinate systems are built up. Important expressions for Cartesian coordinates are shown in Table (4).

When using 1D or 2D Cartesian coordinates, you always drop the last axes. Thanks to rotational symmetry, you can always relabel the axes so that 2D uses (x, y) and 1D uses (x).

quantity		expression		
name	symbol	1D	2D	3D
coordinates	$\boldsymbol{r}$	(x)	(x, y)	(x, y, z)
line element	$d\boldsymbol{\ell}$	$dx\,\hat{\boldsymbol{x}}$	$dx\,\hat{\boldsymbol{x}} + dy\,\hat{\boldsymbol{y}}$	$dx\,\hat{\boldsymbol{x}} + dy\,\hat{\boldsymbol{y}} + dz\,\hat{\boldsymbol{z}}$
volume element	dV	dx	$dx\,dy$	$dx\,dy\,dz$
gradient	$\boldsymbol{\nabla} f$	$\left(\frac{\partial f}{\partial x}\right)\hat{\boldsymbol{x}}$	$\left(\frac{\partial f}{\partial x}\right)\hat{\boldsymbol{x}} + \left(\frac{\partial f}{\partial y}\right)\hat{\boldsymbol{y}}$	$\left(\frac{\partial f}{\partial x}\right)\hat{\boldsymbol{x}} + \left(\frac{\partial f}{\partial y}\right)\hat{\boldsymbol{y}} + \left(\frac{\partial f}{\partial z}\right)\hat{\boldsymbol{z}}$
divergence	$\boldsymbol{\nabla} \cdot \boldsymbol{g}$	$\frac{\partial g_x}{\partial x}$	$\frac{\partial g_x}{\partial x} + \frac{\partial g_y}{\partial y}$	$\frac{\partial g_x}{\partial x} + \frac{\partial g_y}{\partial y} + \frac{\partial g_z}{\partial z}$

Table 4: Important expressions for Cartesian coordinates.

II.B. Cylindrical Coordinates

The next step up in complexity is cylindrical coordinates. Important expressions for cylindrical coordinates are shown in Table (5).

When we refer to "1D cylindrical coordinates", the usual meaning is (ρ): functions depend on the distance from the z axis only. One could, in principle, have other 1D cylindrical coordinate systems: (ϕ) or (z). The (ϕ) coordinate system is rarely, if ever, seen. The (z) coordinate system simply reduces to 1D Cartesian coordinates. Thus we only need the radial 1D cylindrical coordinate system (ρ).

Cylindrical also has three different possible 2D coordinate systems: (ρ, ϕ), (ρ, z), and (ϕ, z). The

quantity		expression	
name	symbol	1D radial	2D polar
coordinates	$\boldsymbol{r}$	(ρ)	(ρ, ϕ)
line element	$d\boldsymbol{\ell}$	$d\rho\,\hat{\boldsymbol{\rho}}$	$d\rho\,\hat{\boldsymbol{\rho}} + \rho\,d\phi\,\hat{\boldsymbol{\phi}}$
volume element	dV	$2\pi\,\rho\,d\rho$	$\rho\,d\rho\,d\phi$
gradient	$\boldsymbol{\nabla} f$	$\left(\frac{\partial f}{\partial \rho}\right)\hat{\boldsymbol{\rho}}$	$\left(\frac{\partial f}{\partial \rho}\right)\hat{\boldsymbol{\rho}} + \left(\frac{1}{\rho}\frac{\partial f}{\partial \phi}\right)\hat{\boldsymbol{\phi}}$
divergence	$\boldsymbol{\nabla} \cdot \boldsymbol{g}$	$\frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho g_\rho\right)$	$\frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho g_\rho\right) + \frac{1}{\rho}\frac{\partial g_\phi}{\partial \phi}$
name	symbol	2D axisymmetric	3D
coordinates	$\boldsymbol{r}$	(ρ, z)	(ρ, ϕ, z)
line element	$d\boldsymbol{\ell}$	$d\rho\,\hat{\boldsymbol{\rho}} + dz\,\hat{\boldsymbol{z}}$	$d\rho\,\hat{\boldsymbol{\rho}} + \rho\,d\phi\,\hat{\boldsymbol{\phi}} + dz\,\hat{\boldsymbol{z}}$
volume element	dV	$2\pi\,\rho\,d\rho\,dz$	$\rho\,d\rho\,d\phi\,dz$
gradient	$\boldsymbol{\nabla} f$	$\left(\frac{\partial f}{\partial \rho}\right)\hat{\boldsymbol{\rho}} + \left(\frac{\partial f}{\partial z}\right)\hat{\boldsymbol{z}}$	$\left(\frac{\partial f}{\partial \rho}\right)\hat{\boldsymbol{\rho}} + \left(\frac{1}{\rho}\frac{\partial f}{\partial \phi}\right)\hat{\boldsymbol{\phi}} + \left(\frac{\partial f}{\partial z}\right)\hat{\boldsymbol{z}}$
divergence	$\boldsymbol{\nabla} \cdot \boldsymbol{g}$	$\frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho g_\rho\right) + \frac{\partial g_z}{\partial z}$	$\frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho g_\rho\right) + \frac{1}{\rho}\frac{\partial g_\phi}{\partial \phi} + \frac{\partial g_z}{\partial z}$

Table 5: Important expressions for cylindrical coordinates. See the text for detailed discussion, particularly of the volume element dV. The "1D polar" coordinate system really exists in Euclidean 2-space with an imposed symmetry over ϕ, and the "2D axisymmetric" coordinate system really exists in Euclidean 3-space with an imposed symmetry over ϕ. By contrast, the "2D polar" coordinate system is most easily thought of as existing in Euclidean 2-space.

first, (ρ, ϕ), is the typical polar coordinates (see Section (II.D) for a discussion on polar coordinates and why they live here). This is a very common case and will be implemented. The second (ρ, z) is the typical axisymmetric coordinates, and it will also be implemented. The third, (ϕ, z), is unusual and will not be implemented.

There is only one 3D cylindrical coordinate system: (ρ, ϕ, z), which will also be implemented.

When eliminating the z axis to reduce from (ρ, ϕ, z) coordinates to (ρ, ϕ) coordinates, we simply imagine that we are in Euclidean 2-space rather than Euclidean 3-space. Like in Cartesian coordinates, all references to z simply disappear. However, when eliminating a rotational coordinate such as ϕ, we cannot do the same thing. The axisymmetric cylindrical coordinate system (ρ, z) does not have any reference to ϕ, but we still have to consider it as existing in Euclidean 3-space with an imposed symmetry. The (ρ, z) coordinates define a plane in Euclidean 3-space, and the symmetry over ϕ means that this plane is swept around the z axis. That third coordinate still exists, but is ignorable due to symmetry. But we need to take care when constructing the volume element. In 3D

cylindrical coordinates, the volume element is $dV_{3D} = \rho\, d\rho\, d\phi\, dz$. When we go to axisymmetric cylindrical coordinates, instead of dropping terms from the volume element, we now integrate over ϕ to capture the full volume in Euclidean 3-space. This integration is what yields the 2π seen in the volume element for axisymmetric coordinates in Table (5), as well as the 2π in radial coordinates.

When eliminiating the ϕ axis, there are two main conventions for integrating over ϕ. The first is to integrate the volume element from 0 to 2π (the full rotation), as discussed in the previous paragraph. This is the convention used in Table (5) to express the volume element. The alternate convention is to integrate over a small range, typically one radian. In this case, we have $dV = \rho\, d\rho\, dz$, which is actually the volume-per-radian. This is done to eliminate the prefactor (hence why unity is chosen for the integration; working in degrees one might integrate over one degree instead of one radian). To recover the total amount of something in that volume, you would need to multiply by the true range of ϕ, which is 2π. Thus, when using this second convention there is typically a step later in the process that multiplies volumetric quantities by 2π (sometimes only for output to the user or for communication with other packages), which brings this effectively back into line with the first convention. This second convention gives rise to a geometry factor, which is discussed in more detail in Section (III).

II.C. Spherical Coordinates

The next step up in complexity is spherical coordinates. Important expressions for spherical coordinates are shown in Table (6).

quantity		**expression**	
name	symbol	1D radial	3D
coordinates	$\boldsymbol{r}$	(r)	(r, θ, ϕ)
line element	$d\boldsymbol{\ell}$	$dr\,\hat{\boldsymbol{r}}$	$dr\,\hat{\boldsymbol{r}} + r\,d\theta\,\hat{\boldsymbol{\theta}} + r\sin(\theta)\,d\phi\,\hat{\boldsymbol{\phi}}$
volume element	dV	$4\pi\, r^2\, dr$	$r^2 \sin(\theta)\, dr\, d\theta\, d\phi$
gradient	$\boldsymbol{\nabla} f$	$\left(\frac{\partial f}{\partial r}\right)\hat{\boldsymbol{r}}$	$\left(\frac{\partial f}{\partial r}\right)\hat{\boldsymbol{r}} + \left(\frac{1}{r}\frac{\partial f}{\partial \theta}\right)\hat{\boldsymbol{\theta}} + \left(\frac{1}{r\sin(\theta)}\frac{\partial f}{\partial \phi}\right)\hat{\boldsymbol{\phi}}$
divergence	$\boldsymbol{\nabla} \cdot \boldsymbol{g}$	$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 g_r\right)$	$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 g_r\right) + \frac{1}{r\sin(\theta)}\frac{\partial}{\partial \theta}\left(\sin(\theta) g_\theta\right) + \frac{1}{r\sin(\theta)}\frac{\partial g_\phi}{\partial \phi}$

Table 6: Important expressions for spherical coordinates.

Spherical has three different possible 1D coordinate systems: (r), (θ), and (ϕ). The first, (r), is what is generally meant by "1D spherical coordinates" and will be implemented. The second, (θ), and third, (ϕ), would be very unusual and will not be implemented.

Spherical also has three different possible 2D coordinate systems: (r, θ), (r, ϕ), and (θ, ϕ). The first, (r, θ), is related to 2D axisymmetric cylindrical coordinates: it would be somewhat like using polar coordinates on a plane, and then sweeping that plane around a symmetry axis. However, there is no current need for it so it will not be implemented. The second, (r, ϕ), is related to polar coordinates, but is distinct from the typical polar coordinates. See Section (II.D) for a more detailed

discussion. This will not be implemented. The third, (θ, ϕ), is generally used only in special-purpose applications, such as celestial coordinates, and is not likely to be useful for Portage. Thus we will not currently implement any 2D spherical coordinate systems.

There is only one 3D spherical coordinate system: (r, θ, ϕ), which will also be implemented. This is the most complex case and introduces issues that complicated coordinate systems will need to address that the other coordinate systems discussed in this document do not need to worry about. Thus it provides a nice example for other, possibly more complicated, coordinate systems.

As discussed in Section (II.B), when eliminating the ϕ axis you cannot conceptualize this as reducing to Euclidean 2-space, but instead as being in Euclidean 3-space with an imposed symmetry. The same is true of reducing over the θ axis. Thus 1D radial spherical coordinates are inherently in Euclidean 3-space, with imposed symmetries such that quantities only depend on the distance from the origin. Additionally, the geometry factor concept now extends over two axes: both θ and ϕ: see Section (III) for more detail.

II.D. Discussion Regarding Polar Coordinates

While at first it seems that polar coordinates could be a reduction of either cylindrical or spherical coordinates, the "usual" polar coordinates are properly thought of as a part of the cylindrical coordinates family. This is due to the fact that spherical coordinates implies a third dimension, which is curved. To see this best, look at the 1D divergence in Cartesian, cylindrical, and spherical coordinates, in which you see a sequence:

$$\text{Cartesian: } \frac{\partial g_x}{\partial x} \qquad \text{cylindrical: } \frac{1}{\rho}\frac{\partial}{\partial \rho}\left(\rho g_\rho\right) \qquad \text{spherical: } \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 g_r\right). \tag{1}$$

You start with effectively x^0, then move to ρ^1, and then to r^2. If polar coordinates were to be considered the 2D version of spherical coordinates, then you would have to deal with one of two problems: either (a) you would be using an unusual polar coordinates with a third implied dimension, which is curved; or (b) your 1D and 3D spherical coordinates would have r^2 terms but your 2D spherical coordinates would be inconsistent and would instead have r^1 terms. Instead, we consider polar coordinates to be a part of the cylindrical coordinates family.

Because of this, we cannot simply say that we have 1D, 2D, and 3D versions of Cartesian, cylindrical, and spherical coordinates. Instead we have two different versions of 2D cylindrical coordinates and no 2D spherical coordinates. Thus an integer template `D` and an enum parameter `CoordSys` (having values `CARTESIAN`, `CYLINDRICAL`, and `SPHERICAL`) are insufficient to specify the coordinate system. Instead, it is necessary to have the `CoordSys` template parameter be more complex, and also to ensure that the dimensionality is consistent with the coordinate system.

III. Geometry Factors

Some codes choose to drop numerical factors off of their volume elements and related quantities in order to ease calculations and avoid duplicating those factors throughout the code. These factors are generally not important: so long as your intensive quantities are correct (in order to ensure that functions such as an equation of state give the proper results), your extensive quantities will all

be different from the "real, physical" answer by the same factor and it all generally washes out internally. However, when outputting physically meaningful values, you need to remember to put the geometry factor back in for your extensive quantities. The most obvious choice of geometry factor can essentially be read off from the volume elements. We call these the "surface" geometry factors because they derive from reducing the rotational coordinates' ranges to get surfaces with unit areas. For cylindrical coordinates that do not explicitly include ϕ, this reduces the range of ϕ to one radian. For 1D spherical coordinates, this reduces the surface to (one steradian) $\times r^2$.

Some codes instead choose to use what we might call the "volume" geometry factors, because they derive from integrating over an entire solid body and removing the numerical factor from those quantities. Thus we see that: a rectangular prism would have a volume of $L\,W\,H$ with no numerical factor, implying a geometry factor of unity; a cylinder would have a volume of $\pi\,R^2\,H$, implying a geometry factor of π; and a sphere would have a volume of $\frac{4}{3}\pi\,R^3$, implying a geometry factor of $\frac{4}{3}\pi$. These "volume" geometry factors are less common, but still occasionally seen.

In contrast, many codes choose not to use a geometry factor at all. This means that every quantity within the code is the real, physically-true value. This is equivalent to having all geometry factors set to unity, and we call this the "physical" geometry factor.

All three geometry factor conventions are summarized in Table (7). Portage prefers to use the "physical" geometry factors to avoid having to worry about this issue and avoid confusion when interfacing with codes that are unfamiliar with the concept of a geometry factor. However, in order to be explicit, Portage does include the geometry factors, but they are always set to unity.

	geometry factor		
coordinate system	surface	volume	physical
3D Cartesian	1	1	1
2D Cartesian	1	1	1
1D Cartesian	1	1	1
3D cylindrical	1	1	1
2D polar cylindrical	1	1	1
2D axisymmetric cylindrical	2π	π	1
1D radial cylindrical	2π	π	1
3D spherical	1	1	1
1D radial spherical	4π	$4\pi/3$	1

Table 7: Common choices for geometry factor. Portage chooses to use the "physical" geometry factor.

When interfacing with other codes, it is better to not put in an explicit, hard-coded conversion factor between codes. Instead, each code should store its own geometry factor and the interface between the two codes should construct the conversion factor from the geometry factors. That way, if one code changes their geometry factor, the interface will continue to work without modifications. If explicit conversion factors are put in place, then changing the geometry factor in either

code would require knowing to look for and adjust the conversion factor to maintain consistency.

When passing data between two different codes, whether or not to apply the geometry factor depends on whether the quantity is intensive or extensive. Intensive quantities should be passed without any change. Extensive quantities should be converted by multiplying by the source code geometry factor to get the physically-correct value, and then dividing by the destination code geometry factor to be consistent with the assumptions of the destination code.

IV. Required Data and Operations

Each coordinate system needs to provide certain data and operations, explained here. For clarity, we will introduce a new notation: we will label the coordinate axes c_1, c_2, and c_3. Most of Portage assumes that these are x, y, and z (Cartesian coordinates), but then the coordinate system logic will have to modify certain expressions.

IV.A. Geometry Factor

As discussed in Section (III), each coordinate system has a geometry factor associated with it. Despite Portage choosing to have geometry factors of unity, we want this to be explicit (and we also want this to be easy to change should a decision be made in the future to use a different geometry factor). Thus each coordinate system must provide the geometry factor.

As the inverse geometry factor is also frequently needed, and because divisions are more expensive than multiplications, each coordinate system is expected to provide an inverse geometry factor (1 / geometry factor) that is precomputed a single time to avoid the excess divisions.

As both the geometry factor and its inverse are known ahead of time, they can be made into compile-time constants.

IV.B. Verify Coordinate System

This routine is designed to verify any assumptions that are made regarding this coordinate system. Currently it only verifies the dimensionality. That is, if the coordinate system requested is 2D polar cylindrical, we need to ensure that the dimensionality is set to 2. But it provides a useful place to add any other necessary checks. It is recommended that any checks be done using static asserts in order to catch issues at compile time as much as possible, but this is not a requirement.

IV.C. Modify Gradient

The expression for the gradient differs between coordinate system. As Portage was originally written under the assumption of Cartesian coordinates (and Cartesian is the simplest case), we chose to leave the Cartesian calculations in place and provide a routine that modifies a Cartesian-like gradient to be appropriate for the coordinate system that is actually being used.

To clarify, let us take the example of 3D cylindrical coordinates. The gradient will be calculated

as if the coordinate system is Cartesian, which will give us

$$\left(\frac{\partial f}{\partial c_1}\right)\hat{\boldsymbol{c}}_1 + \left(\frac{\partial f}{\partial c_2}\right)\hat{\boldsymbol{c}}_2 + \left(\frac{\partial f}{\partial c_3}\right)\hat{\boldsymbol{c}}_3 = \left(\frac{\partial f}{\partial \rho}\right)\hat{\boldsymbol{\rho}} + \left(\frac{\partial f}{\partial \phi}\right)\hat{\boldsymbol{\phi}} + \left(\frac{\partial f}{\partial z}\right)\hat{\boldsymbol{z}}, \tag{2}$$

However, the correct expression is

$$\left(\frac{\partial f}{\partial \rho}\right)\hat{\boldsymbol{\rho}} + \left(\frac{1}{\rho}\frac{\partial f}{\partial \phi}\right)\hat{\boldsymbol{\phi}} + \left(\frac{\partial f}{\partial z}\right)\hat{\boldsymbol{z}}. \tag{3}$$

Therefore, we need to divide the $\hat{\boldsymbol{\phi}}$ (a.k.a. $\hat{\boldsymbol{c}}_2$) term by ρ (a.k.a. c_1) in order to correct the gradient for 3D cylindrical coordinates.

This correction to the gradient requires a reference point. As a major use of the gradient is for second-order interpolation, the most logical point to choose is the cell centroid (second-order interpolation is only achieved with linear reconstructions if the reconstruction is constrainted to have the average value in the cell at the cell centroid). However, the routine to modify the gradient will take an arbitrary point should the calling code know that a different reference point is more appropriate in that particular instance.

IV.D. Modify Line Element

The line element modifications are very analogous to those for the gradient. It is assumed that Portage will perform the Cartesian calculations, but then the line element needs to be modified to be appropriate for the coordinate system being used. The modification for the line element needs a reference point. A significant use of line elements is in the second-order interpolation, so the most logical reference point is the cell centroid. However, in other contexts, it is up to the developer to make the most logical choice for the reference point given the current use-case.

IV.E. Moments

Moments are somewhat complex, so Portage requires several operations here. The complexity largely arises from the fact that a moment is an integral over a region, but Portage is intended to handle many different kinds of regions and thus cannot know at compile time the shape to use for moments. Using a different shape would lead to a different expression for the moment.

IV.E.1. Modify Moments (Shift)

For an arbitrary cell shape, we can take advantage of certain properties of moments to shift the moments around within the moments list. This has the advantage that the cell shape need not be known, but it has the disadvantage that extra moments need to be computed and then thrown away.

The expressions for the moment-shift method are summarized in Table (8). A more detailed discussion can be found in Appendix (A).

IV.E.2. Moment Shift Amount

When shifting the moments, we lose orders of moments. For example, if we are provided with all moments up through third order, and we are in 2D cylindrical coordinates where we have to

moment shifts	
coordinate system	shift equation
3D Cartesian	$\mathcal{M}[i][j][k] = \frac{1}{G} S[i][j][k]$
2D Cartesian	$\mathcal{M}[i][j] = \frac{1}{G} S[i][j]$
1D Cartesian	$\mathcal{M}[i] = \frac{1}{G} S[i]$
3D cylindrical	$\mathcal{M}[i][j][k] = \frac{1}{G} S[i+1][j][k]$
2D polar cylindrical	$\mathcal{M}[i][j] = \frac{1}{G} S[i+1][j]$
2D axisymmetric cylindrical	$\mathcal{M}[i][k] = \frac{2\pi}{G} S[i+1][k]$
1D radial cylindrical	$\mathcal{M}[i] = \frac{2\pi}{G} S[i+1][j][k]$
3D spherical	$\mathcal{M}[i][j][k] = \frac{1}{G} \sum_{n=0}^{\infty} \left(\frac{(-1)^n}{(2n)!} S[i+2][j+2n][k] \right)$
1D radial spherical	$\mathcal{M}[i] = \frac{4\pi}{G} S[i+2][j][k]$

Table 8: Summary of the expressions for the moment-shift method. The geometry factor is given as G. See Section (III) for more detail about the geometry factor. See Appendix (A) for more detail about where these expressions come from, as well as for discussion of the complications (and possible approaches) for 3D spherical coordinates.

shift the ρ exponent by one, we lose one order of moments and are left with only up through the second-order moments. An example of how this works is shown in Table (9).

Thus we need to know how many orders of moments we lose when applying this shift. For all Cartesian coordinate systems, the shift amount is zero. For all cylindrical coordinate systems, the shift amount is one. For all spherical coordinate systems, the shift amount is two[4].

[4]For 3D spherical coordinates, you need an infinite number of moments in order to calculate the new moments exactly (see Appendix (A)). However, you *lose* two orders of moments, so the moment shift is not infinity, but is two.

integral		moment	
order	exponents	order	exponents
0	S[0][0]	—	thrown out
1	S[1][0]	0	$\mathcal{M}$[0][0]
1	S[0][1]	—	thrown out
2	S[2][0]	1	$\mathcal{M}$[1][0]
2	S[1][1]	1	$\mathcal{M}$[0][1]
2	S[0][2]	—	thrown out
3	S[3][0]	2	$\mathcal{M}$[2][0]
3	S[2][1]	2	$\mathcal{M}$[1][1]
3	S[1][2]	2	$\mathcal{M}$[0][2]
3	S[0][3]	—	thrown out

Table 9: Example of applying moment-shift method, demonstrating that for 2D cylindrical coordinates one order of moments is lost.

IV.E.3. Modify Moments (Axis-Aligned Boxes)

As an optimization, we know that some meshes will consist only of axis-aligned boxes. The moments for axis-aligned boxes are relatively straightforward to compute[5] and code up explicitly. This removes the need to compute extra moments, as we can simply modify the existing moments based on knowing the bounds of the axis-aligned box. An additional advantage of this method is that it can explicitly be computed for every coordinate system and thus even 3D spherical coordinates work if the mesh is composed of axis-aligned boxes and the moment-shift method is avoided. These methods do need to be explicitly provided with the bounds of the axis-aligned box, so they require not just the extra assumption regarding the shape but also additional information. That means they cannot be drop-in replacements for the moment-shift methods described above.

Because the expressions need to be explicitly calculated and then coded up, the current implementation only provides zeroth (volume) and first moments. If desired, some cleverness might be able to create general expressions that apply for all moments[6].

Any shape that is known a priori can be similarly computed, and then an analytic expression hard-coded in as an optimization. The axis-aligned boxes methods provide a model for this.

V. Implementation

As discussed in Section (II.D), we cannot simply represent the coordinate system with the pair of an integer template and an enum template that can take the values `CARTESIAN`, `CYLINDRICAL`, and `SPHERICAL`. Instead we need to explicitly call out the coordinate systems more specifically

[5]Because your favorite symbolic math tool can spit these out rather trivially, or you can work them by hand yourself, I'll avoid making this document even longer than it already is by showing derivations or results.

[6]In other words: I don't currently see a reason to put in this much effort, so I leave that as an exercise to the ambitious reader who may want to expand on the current work.

and verify that the dimensionality is consistent with the selected coordinate system.

The initial thought was to use enum template parameters that specify the coordinate system and modify each routine to be aware of the coordinate system. However, there began to be excessive code duplication, so we wanted to collect together the same operations into a single routine that could be called instead of repeating the same logic in multiple places. Using an enum made this process awkward: logic was organized by operation rather than by coordinate system, so every operation needed to be updated for every coordinate system, making it easy to miss an operation when adding or removing a new coordinate system. Additionally, data had to be managed by using the enum as an index to an array. While this is perfectly legal according to the standard and in common uses in some code bases, it is non-obvious to developers not used to this practice. The enum implementation felt unnecessarily fragile.

The second implementation was to use structs. Each coordinate system provided a struct with the static data and operations. Routines that need to be aware of the coordinate system are templated on the coordinate system, but we never need to actually instantiate a coordinate system object because of the decision to have all data and operations be static. Thus all information for a given coordinate system is collected together into a single struct. Removing a coordinate system is as simple as deleting the struct. In order to add a new coordinate system, it is easy to copy the struct for an existing coordinate system and use that as a model for the new coordinate system.

VI. Modifications to Existing Code

A major goal of implementing non-Cartesian coordinate systems in Portage was to minimize (if not entirely avoid) changes to the interface. Thus the coordinate systems were implemented as structs, as discussed in Section (V), and a default coordinate system (Cartesian coordinates for backwards-compatibility) was defined through a typedef[7]. We went through several design iterations, working through issues with default templates and partial template specifications and found that we were able to design the interface so that users of Portage should not need to change their code, and they will get the default coordinate system and everything will work as before.

The first thing that needed to be changed to accomodate new coordinate systems was the second-order interpolator (and thus all interpolators in order to maintain a uniform interface). The second-order interpolator uses a reconstruction of the form

$$f(\boldsymbol{r}) = f(\boldsymbol{r_c}) + \boldsymbol{\nabla} f \cdot d\boldsymbol{\ell}, \tag{4}$$

where r_c is the position of the cell centroid. In 3D Cartesian coordinates, this can be expanded as

$$\begin{aligned} f(\boldsymbol{r}) &= f(\boldsymbol{r_c}) + \boldsymbol{\nabla} f \cdot d\boldsymbol{\ell} && (5)\\ &= f(\boldsymbol{r_c}) + \frac{\partial f}{\partial x}\, dx + \frac{\partial f}{\partial y}\, dy + \frac{\partial f}{\partial z}\, dz && (6)\\ &= f(\boldsymbol{r_c}) + \frac{\partial f}{\partial c_1}\, dc_1 + \frac{\partial f}{\partial c_2}\, dc_2 + \frac{\partial f}{\partial c_3}\, dc_3. && (7) \end{aligned}$$

[7] In the world of templates, a typedef is analogous to having a named constant variable to hold a value, rather than hard-coding that value in multiple places throughout the code. Named constants are always preferred as they provide clarity to the origins of the values in a code and allow for early assumptions to be easily changed as the code evolves.

However, in other coordinate systems, both the gradient and the line element change form. It turns out that, in all coordinate system considered here, the changes to the gradient and line element cancel each other out. Therefore, we could have left the interpolator as-is (that is, using Equation (7) for all coordinate systems). However, this relies on a special cancellation that would need to be proven rigorously for all coordinate systems[8]and then clearly documented. Instead, we modify both the gradient and the line element for the new coordinate system for clarity and consistency, which means that the interpolator needs to be aware of the coordinate system.

The only caveat to "not changing the interface" is if the user is using template templates. If they use an interpolator template template rather than a fully-specialized class template, then they will have to update their template templates to include an additional class for the coordinate system.

Any part of Portage that does not include the coordinate system template parameter is assumed to either (a) not need any changes from the coordinate system; or (b) only be valid for Cartesian coordinates and not yet be available for use with other coordinate systems. We should work through the entire Portage code base, identify the pieces that need to be modified for the coordinate system, and mark them in some way in order to distinguish between these two cases. This could be done, for example, by templating that piece of code on the coordinate system, but then putting in a static assertion that the coordinate system must be Cartesian. That will warn developers and users that the requested feature is not yet ready for use with non-Cartesian coordinate systems.

A. Appendix: Details of the Moment Shift Method

Portage is currently able to provide the Cartesian moments. A moment is defined as

$$M_q = \iiint_V q \, dV, \tag{8}$$

where q is the quantity that you are taking the moment of, and V is the volume of the region over which you are taking the moment. Thus we might ask for the second moment with respect to xy:

$$M_{xy} = \iiint_V x \, y \, dV. \tag{9}$$

Portage currently only needs moments of powers of the coordinate axes. So we use the the notations from Section (IV), and define (e.g., in 3D)

$$\mathcal{M}[i][j][k] \equiv \iiint_V c_1^i \, c_2^j \, c_3^k \, dV, \tag{10}$$

and

$$S[i][j][k] \equiv \iiint_V c_1^i \, c_2^j \, c_3^k \, dc_1 \, dc_2 \, dc_3, \tag{11}$$

where c_1, c_2, and c_3 are the coordinate axes. Portage originally only used Cartesian coordinates, where $dV = dx\,dy\,dz = dc_1\,dc_2\,dc_3$, so that $\mathcal{M}[i][j][k] = S[i][j][k]$. Because of this, all moments

[8]I believe this is true for all orthogonal curvilinear coordinate systems (you can look up Lamé coefficients and their use for general orthogonal curvilinear coordinate systems in Euclidean space), but I have not demonstrated it rigorously as it was not necessary to prove for the purposes of this effort. I have not even considered the question of whether or not this is true for non-orthogonal curvilinear coordinate systems.

are currently computed according to the equation for $S[i][j][k]$. However, other coordinate systems have additional terms in their volume elements, so that this equality doesn't hold and we need to determine the relation between what we want (the moment $\mathcal{M}$) and what we have (the integral S).

With most of our coordinate systems, the volume element will only add factors of the coordinate axes. For example, the second moment in cylindrical coordinates with respect to $\rho\phi$ is

$$M_{\rho\phi} = \iiint \rho\,\phi\,dV = \iiint \rho^2\,\phi\,d\rho\,d\phi\,dz. = \iiint c_1^2\,c_2\,dc_1\,dc_2\,dc_3. \tag{12}$$

Notice that the volume element increments the exponent on $c_1 = \rho$, so that

$$M_{\rho\phi} = M[1][1][0] = S[2][1][0]. \tag{13}$$

There are also corrections for the geometry factor, G. As moments are extensive quantities, they need to be divided by the geometry factor. This can be seen by considering the zeroth and first moments. The zeroth moment is the volume, and we have already seen that the volume is modified by the geometry factor. The first moments are equivalent to the centroid multiplied by the volume, and we've already seen that the volume is modified by the geometry factor. We can extend this logic and see that all moments, as extensive quantities, need to be divided by the geometry factor. Additionally, there is sometimes another term to be included along with the geometry factor. Computing this term correctly, along with the appropriate shifts, may require more careful treatment. The derivation for each coordinate system is given below.

The Cartesian moments are trivial: simply divide by the geometry factor. Thus:

$$\mathcal{M}[i] = \frac{1}{G}\,S[i] \tag{14}$$

$$\mathcal{M}[i][j] = \frac{1}{G}\,S[i][j] \tag{15}$$

$$\mathcal{M}[i][j][k] = \frac{1}{G}\,S[i][j][k] \tag{16}$$

$$\tag{17}$$

The 3D cylindrical moments are

$$\begin{aligned}
\mathcal{M}[i][j][k] &= \frac{1}{G}\iiint_V \rho^i\,\phi^j\,z^k\,dV \\
&= \frac{1}{G}\iiint_V \rho^i\,\phi^j\,z^k\,\rho\,d\rho\,d\phi\,dz \\
&= \frac{1}{G}\iiint_V \rho^{i+1}\,\phi^j\,z^k\,d\rho\,d\phi\,dz \\
&= \frac{1}{G}\iiint_V c_1^{i+1}\,c_2^j\,c_3^k\,dc_1\,dc_2\,dc_3 \\
&= \frac{1}{G}\,S[i+1][j][k].
\end{aligned} \tag{18}$$

In 2D cylindrical polar coordinates, the moments are:

$$\begin{aligned}\mathcal{M}[i][j] &= \frac{1}{G}\iint_V \rho^i\,\phi^j\,dV \\ &= \frac{1}{G}\iint_V \rho^i\,\phi^j\,\rho\,d\rho\,d\phi \\ &= \frac{1}{G}\iint_V \rho^{i+1}\,\phi^j\,d\rho\,d\phi \\ &= \frac{1}{G}\iint_V c_1^{i+1}\,c_2^j\,dc_1\,dc_2 \\ &= \frac{1}{G}\,S[i+1][j]. \end{aligned} \tag{19}$$

Recall from Section (II.B) that 2D cylindrical axisymmetric coordinates really exist in Euclidean 3-space with an imposed symmetry such that there is no dependence on ϕ. Thus, we imagine that the 2D "volume" is swept around a full rotation to generate the full 3D volume V_{3D}. The moments are

$$\begin{aligned}\mathcal{M}[i][j] &= \frac{1}{G}\iiint_{V_{3D}} \rho^i\,z^j\,dV \\ &= \frac{1}{G}\iiint_{V_{3D}} \rho^i\,z^j\,\rho\,d\rho\,d\phi\,dz \\ &= \frac{1}{G}\int_0^{2\pi} d\phi \iint_V \rho^{i+1}\,z^j\,d\rho\,dz \\ &= \frac{2\pi}{G}\iint_V c_1^{i+1}\,c_2^k\,dc_1\,dc_2 \\ &= \frac{2\pi}{G}\,S[i+1][j]. \end{aligned} \tag{20}$$

Notice that, in this coordinate system, $z = c_2$ rather than $z = c_3$, because the ϕ coordinate is integrated out. In order to verify this expression for the moments, consider using the "physical" geometry factor $G = 1$ and computing the volume: $V = \mathcal{M}[0][0]$. In this case, we should recover $V = \iiint dV$, which we do. Similarly, if we used the "surface" geometry factor $G = 2\pi$, we should get the volume per radian, or $V = \iiint dV/2\pi$, which we do.

In 1D cylindrical radial coordinates, the coordinate system exists in Euclidean 2-space, and we imagine that the 1D "volume" is swept around a full rotation to generate the 2D "volume" V_{2D}.

The moments are

$$\begin{aligned}
\mathcal{M}[i] &= \frac{1}{G} \iint_{V_{2D}} \rho^i \, dV \\
&= \frac{1}{G} \iint_{V_{2D}} \rho^i \, \rho \, d\rho \, d\phi \\
&= \frac{1}{G} \int_0^{2\pi} d\phi \int_V \rho^{i+1} \, d\rho \\
&= \frac{2\pi}{G} \int_V c_1^{i+1} \, dc_1 \\
&= \frac{2\pi}{G} S[i+1]. \qquad (21)
\end{aligned}$$

As discussed in Section (II.C), 1D spherical radial coordinates exist in Euclidean 3-space, so we imagine that the 1D "volume" is swept around the complete sphere to generate the full 3D volume V_{3D}. The moments are

$$\begin{aligned}
\mathcal{M}[i] &= \frac{1}{G} \iiint_{V_{3D}} r^i \, dV \\
&= \frac{1}{G} \iiint_{V_{3D}} r^i \, r^2 \, \sin\theta \, dr \, d\theta \, d\phi \\
&= \frac{1}{G} \int_0^{\pi} \sin\theta \, d\theta \int_0^{2\pi} d\phi \int_V r^{i+2} \, dr \\
&= \frac{4\pi}{G} \int_V c_1^{i+2} \, dc_1 \\
&= \frac{4\pi}{G} S[i+2]. \qquad (22)
\end{aligned}$$

The 3D spherical moments are

$$\begin{aligned}
\mathcal{M}[i][j][k] &= \frac{1}{G} \iiint_V r^i \, \theta^j \, \phi^k \, dV \\
&= \frac{1}{G} \iiint_V r^i \, \theta^j \, \phi^k \, r^2 \, \sin\theta \, dr \, d\theta \, d\phi \\
&= \frac{1}{G} \iiint_V r^{i+2} \, \theta^j \, \phi^k \, \sin\theta \, dr \, d\theta \, d\phi. \qquad (23)
\end{aligned}$$

Unfortunately, this includes a transcendental term: $\sin\theta$. Using a Taylor expansion:

$$\begin{aligned}
\mathcal{M}[i][j][k] &= \frac{1}{G}\iiint_V r^{i+2}\,\theta^j\,\phi^k\,\sin\theta\,dr\,d\theta\,d\phi \\
&= \frac{1}{G}\iiint_V r^{i+2}\,\theta^j\,\phi^k\left(\sum_{n=0}^{\infty}\frac{(-1)^n\,\theta^{2n}}{(2n)!}\right)dr\,d\theta\,d\phi \\
&= \frac{1}{G}\iiint_V r^{i+2}\left(\sum_{n=0}^{\infty}\frac{(-1)^n\theta^{j+2n}}{(2n)!}\right)\phi^k\,dr\,d\theta\,d\phi \\
&= \frac{1}{G}\sum_{n=0}^{\infty}\left(\frac{(-1)^n}{(2n)!}\iiint_V r^{i+2}\,\theta^{j+2n}\,\phi^k\,dr\,d\theta\,d\phi\right) \\
&= \frac{1}{G}\sum_{n=0}^{\infty}\left(\frac{(-1)^n}{(2n)!}\,S[i+2][j+2n][k]\right). \qquad (24)
\end{aligned}$$

This expression is only accurate if we perform the full infinite sum. However, it can be used as the basis for an approximation: For every 3D spherical moment you want to compute, use as many S integrals in the summation as are available. More moments should make this more accurate. Alternately, the moment-shift method can simply generate an error for 3D spherical coordinates.

BKK

Copy: Joann Campbell, XCP-2, jomc@lanl.gov
Christopher Werner, XCP-2, cwerner@lanl.gov