Engineering with Computers (2002) 18: 286–294

Engineering with Computers

On Object-Oriented Frameworks and Coordinate Free Formulations of PDEs

K. Åhlander[1], M. Haveraaen[2] and H. Munthe-Kaas[2]

[1]Department of Information Technology, University of Uppsala, Uppsala, Sweden;
[2]Institute of Informatics, University of Bergen, Bergen, Norway

Abstract. *An object-oriented (OO) framework for Partial Differential Equations (PDEs) provides software abstractions for numerical simulation of PDEs. The design of such frameworks is not trivial, and the outcome of the design is highly dependent on which mathematical abstractions one chooses to support. In this paper, coordinate free abstractions for PDEs are advocated. The coordinate free formulation of a PDE hides the underlying coordinate system. Therefore, software based on these concepts has the prospect of being more modular, since the PDE formulation is separated from the representation of the coordinates. Use of coordinate free methods in two independent OO frameworks are presented, in order to exemplify the viability of the concepts. The described applications simulate seismic waves for various classes of rock models and the incompressible Navier-Stokes equations on curvi-linear grids, respectively. In both cases, the possibility to express the equations in a domain independent fashion is crucial. Similarities and differences between the two coordinate free frameworks are discussed. A number of places where such frameworks should be designed for modification is identified. This identification is of interest both for framework developers and for tentative framework users.*

Keywords. Computations on manifolds; Coordinate free numerics; Object-oriented frameworks; Partial differential equations

1. Introduction

The numerical solution of Partial Differential Equations (PDEs) is a major part of the solution process for many engineering problems. Many, often conflicting, demands are put on PDE simulation software. The software must be easy to use, fast, flexible, reliable, etc. It is unlikely that we will obtain one simulation package ideal for all different kinds of PDEs, but there exist several projects with the aim of providing simulation capabilities to a large set of PDE problems, for instance Cogito [1], Diffpack [2], Overture [3], Pooma [4] and Pellpack [5]. The simulation capabilities are usually provided via an Object-Oriented (OO) framework, understood as a collection of related classes that may be combined to solve a particular problem [6].

However, the design of OO frameworks for PDEs is not a trivial problem, and there are many issues where different approaches yield different solutions. In this paper, we discuss one of the most basic questions regarding the design of PDE simulation software: Which mathematical abstractions shall be the foundation for an OO PDE framework?

Traditionally, applied mathematicians have started program development from a given PDE in component form. This formulation contains implicit information, for instance regarding the coordinate system. The component form may also be based upon various simplifications, which are then implicitly hidden in the equations. As a consequence, it may be difficult to change coordinate systems or to model the same PDE under circumstances where the original simplifications can not be applied [7].

We argue that it is important to base the software abstractions on the original, coordinate free formulation of the PDE. This formulation captures the essence of the PDE and is therefore more robust. The choice of coordinate system, as well as the introduction of various simplifications, then enter into software design in a later stage of the analysis process. Examples of when this principle shows its strength with respect to code are:

- When the physics suggests a particular coordinate system. One example is when cylindrical coordinates simplify problems with a rotational symmetry.

Correspondence and offprint requests to: K Ahlander, Dept of Information Technology, University of Uppsala, Box 337, SE-75105 Uppsala, Sweden. E-mail: krister@tdb.uu.se

- When the numerical method maps the computational grid to the actual geometry. This situation is typical when finite differences are used to simulate PDEs that are expressed on curvi-linear grids.

In this paper we concentrate on coordinate free abstractions for *space operations*, such as spatial derivations and spatial integration. In non-relativistic engineering applications we may consider the computational domain as factorized in a product of a time direction and space directions, which may be treated independently of each other. The time stepping procedures we employ in the examples are simple traditional coordinate based algorithms, such as Euler updates and leapfrog. It should be noted that during the recent years a substantial effort has been done in understanding also time stepping algorithms in a coordinate free framework. These algorithms are based on (coordinate free) Lie group actions as the basis for updating the numerical solution in time. We refer the readers elsewhere [8, 9] and the references therein for an extensive discussion of this topic.

The outline of this paper is as follows. In Section 2, we motivate the coordinate free approach. In Section 3, we discuss some main abstractions for a coordinate free approach. In Section 4, we present two sample PDE problems from different areas, where a coordinate free approach has been beneficial, and we relate the coordinate free abstractions to the setting of OO frameworks. In Section 5, we summarize our findings.

2. Component Form versus Coordinate Free Form

In this section, we highlight the difference between the component form and the coordinate free form of a PDE, and we show what this difference implies for the PDE software design. As an illuminating example, we use a simple heat equation:

$$u_t = c\ \triangle u. \tag{1}$$

Here, u is the temperature, c is a given coefficient, and the derivatives u_t and $\triangle u$ denote time derivative and Laplacian spatial derivative[1] of the temperature, respectively. In this simplified example, we assume that appropriate initial conditions and boundary conditions are given, and we focus on how the equation may be solved in various geometries. First, we consider geometries according to Fig. 1.

To solve (1) on a three-dimensional cube (Fig. 1a), it is convenient to write the equation in component form as

$$u_t = c(u_{xx} + u_{yy} + u_{zz}), \tag{2}$$

where u_{xx} denotes second derivative wrt x etc. This example problem is readily solved. However, if the problem is such that the values are independent of z, it may be modeled in two dimensions (Fig. 1b), and we simplify the equation:

$$u_t = c(u_{xx} + u_{yy}). \tag{3}$$

If we, in our example, use finite differences in order to discretize in space, it is trivial to solve either of the Eqs (2) and (3). However, when comparing Eqs (2) and (3), we notice that the same equation actually is written in two different ways, depending basically on the choice of coordinate system. This is a significant disadvantage of the component form. Therefore, it is better if the numerical simulation is based directly upon equation (1), which is the coordinate free formulation. The computations of the Laplacian will still, of course, be carried out in different ways, but this can be hidden from the equation by having a coordinate free interface to the field u. The advantage of a coordinate free interface is even more significant if we consider a curvi-linear grid, see Fig. 1c. Let x and y be mapped from a logically rectangular grid (cf. Fig. 3) with coordinates r and s:

$$\begin{aligned} x &= x(r, s) \\ y &= y(r, s), \end{aligned}$$

and denote the inverse mappings as

$$\begin{aligned} r &= r(x, y) \\ s &= s(x, y), \end{aligned}$$

The computations of the Laplacian may still use finite differences, but one must take the chain rule into account. Equation (3) becomes

$$\begin{aligned} u_t = c((&r_x^2 + r_y^2)u_{rr} \\ &+ 2(r_x s_x + r_y s_y)u_{rs} + (s_x^2 + s_y^2)u_{ss} \\ &+ (r_{xx} + r_{yy})u_r + (s_{xx} + s_{yy})u_s), \end{aligned} \tag{4}$$

where r_x denote the partial derivative of r with respect to x, etc. When doing simulations on a curvi-linear grid, the component form of the PDE quickly becomes impractical. As yet another complication, consider a geometry which is covered by a

[1] Sometimes, the notation ∇^2 is used for the Laplacian. The notation $\triangle$ is chosen to obtain a clearer notation later when software is discussed.

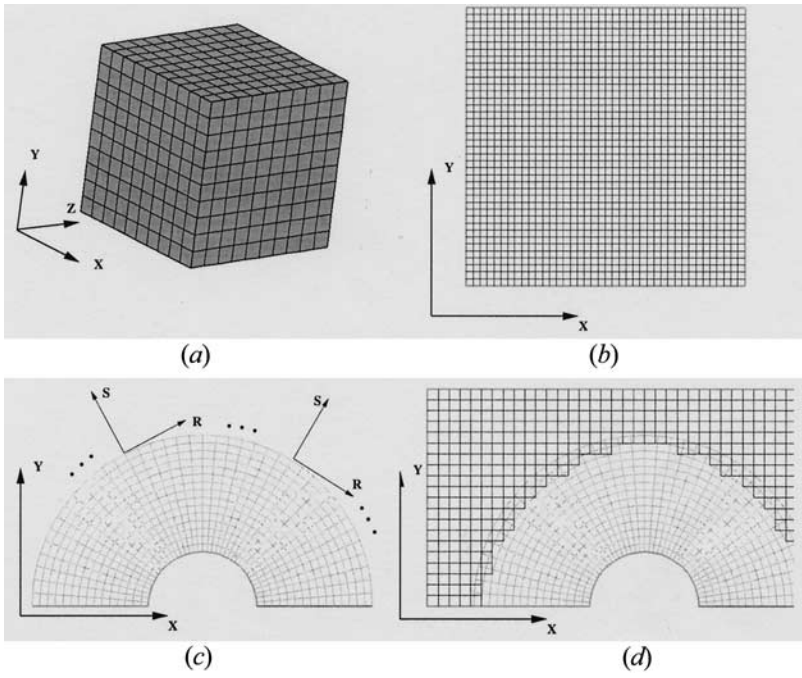


Fig. 1. Sample geometries where a PDE can be simulated: (a) a 3D cube, (b) a 2D square, (c) a curvi-linear grid, and (d) a composite grid.

composite grid (see Fig. 1d). Here, it is natural to use Eq. (3) on one grid and Eq. (4) on the other grid.

Summarizing our example so far, we have shown how the component form of a PDE is sensitive to the choice of coordinate system. The desired situation is to have programming abstractions that provide a coordinate free interface. We would like to be able to express the right hand side of Eq. (1) directly in terms of programming abstractions. If `u` is a `Scalar Field`, it should be possible to simulate Eq. (1) with for instance a Forward Euler step (with time step `dt`), using code similar to:

```
u = u + dt*c*u.Laplacian( );
```

Such a coordinate free interface on curvi-linear grids is actually provided by for instance Overture [3]. Overture embeds a composite overlapping grid with different local coordinate systems into one global coordinate system. It is therefore possible to use partial differential operators like ∂/∂_{xx} and ∂/∂_{yy} independently of the underlying curvilinearity of the manifold. Compound operators like the Laplacian are also available. Overture is discussed further below in Section 4.2. However, a coordinate free interface may be used not only for curvi-linear grids. Consider the geometry of Fig. 2. This shape has a rotational symmetry, and we may simplify the computations using cylindrical coordinates (ρ, ϕ, z). In cylindrical coordinates, $\triangle u$ reads

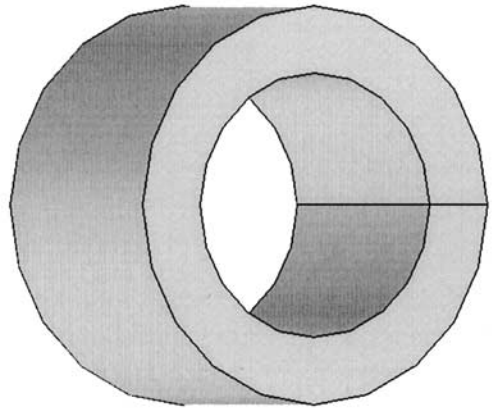


Fig. 2. A rotational shape may be modeled in two dimensions, since everything is constant when the angle changes.

$$\triangle u = u_{\rho\rho} + \frac{1}{\rho} u_{\rho} + \frac{1}{\rho^2} u_{\phi\phi} + u_{zz}.$$

In the case of rotational symmetry, $u_{\phi\phi}$ vanishes. We refer to a coordinate free interface which provides coordinate free implementations in this more general sense for coordinate freeness at the tensor level. Our previous examples with curvi-linear grids have been geared towards finite differences, but the advantages of a coordinate free interface at the tensor level are applicable to other discretization methods as well. The architecture of Sophus [10] has been designed with a coordinate free interface

at the tensor level in mind and it allows us to transparantly handle for instance rotational symmetries. In Grant *et al.* [11], finite elements are used to simulate a wire coating problem with rotational symmetry. Sophus is discussed in more detail in Section 4.1.

3. Abstractions

In this section, we discuss a few main abstractions for coordinate free PDE simulation software. We emphasize the variation points a coordinate free framework should support, both with respect to domain abstractions and coordinate free abstractions on a higher abstraction level. Variation points are places where a framework is designed for modification.

3.1 Domain Abstractions

Coordinate free domain abstractions are perhaps best illustrated by curvi-linear grids which are often used for finite differences. However, the concepts discussed in this section may be applied whenever the manifold in which the PDE evolves, the *physical domain*, differs from the computational domain.

A *manifold* is the mathematical domain needed for differentiation. The simplest example is an open rectangular subset of $\mathcal{R}^n$. The *computational domain* is the manifold where the numerical approximations to the differentiations actually are carried out.

It is often advantageous to use a computational domain which differs from the physical domain, for symmetry reasons (coordinate free tensor level) or for modeling geometry (coordinate free curvi-linear level). The change of coordinates depend on a mapping $\vec{x} = \vec{x}(\vec{r})$ and its inverse mapping $\vec{r} = \vec{r}(\vec{x})$ from the computational domain $\vec{r} = (r,s, \ldots)$, to the physical domain $\vec{x} = (x,y, \ldots)$, see Fig. 3. A given manifold can, in general, not be covered with one single mapping (a chart) from Cartesian space. In pure mathematics, this complication arises in conjunction with the definition of an analytical manifold. The complication is handled (see e.g. Schutz [12]) by introducing an 'atlas' of consistently overlapping local charts over the manifold (see Fig. 4).

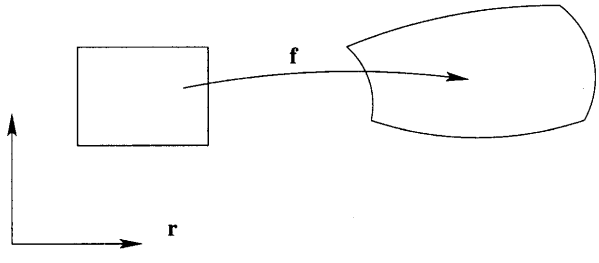


Fig. 3. A mapping (a chart) from a Cartesian space to a physical manifold.

Here, we find it interesting to note the similarities between the concepts of pure mathematics and overlapping composite grids. Overlapping composite grids are used in the finite difference community to handle complex computational domains (cf. Figure 1d). As an example, we refer to the design of computational domains in Overture. Here, the object model is similar to the UML [13] diagram in Fig. 5. The computational domain, a composite grid, consists of several curvi-linear grids, each associated with a mapping. In Overture, `Mapping` is an abstract base class. Various subclasses are provided which cater for different common mappings. The ability to handle various mappings is an important variation point.

Besides the mappings themselves, derivatives of the mappings are required (at every point of the domain), cf. Eq. (4). In software, one has a choice between precomputing these values and computing them on demand. In some cases, the derivatives are known analytically, for instance for cylindrical coordinates, but for other mappings numerical approximations of appropriate accuracy must be performed. In the context of a coordinate free framework, it should be possible to experiment with these choices, since it often is impossible to know beforehand which strategy is the most efficient for a given physical problem on a given platform.

3.2 Coordinate Free Abstractions and Operations

Now, we draw our attention to coordinate free abstractions and operations. We focus on explicit computations and we will use the gradient, ∇, and

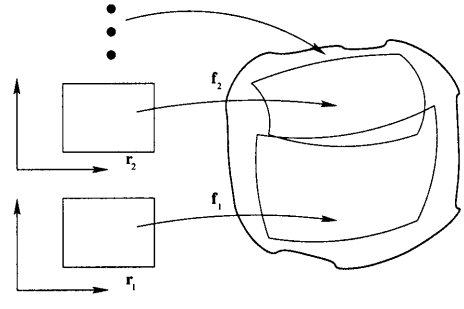


Fig. 4. If the manifold can not be covered with one single mapping, several mappings can be used to create an 'atlas' over the manifold.

Fig. 5. A UML diagram associating grids and mappings in Overture. A composite grid *consists of* (the diamond symbol) *several* (the * symbol) Mapped Grids, each associated with a single mapping.

the Laplacian, $\triangle$, as examples. Other differentiation operations can be treated in a similar fashion.

The Laplacian is a mapping from a scalar field to a scalar field, and the gradient is a mapping from a scalar field to a vector field. We may formalize this as

$$\triangle : \texttt{Scalar Field} \rightarrow \texttt{Scalar Field}$$
$$\nabla : \texttt{Scalar Field} \rightarrow \texttt{Vector Field}.$$

Both scalar fields and vector fields are coordinate free mathematical objects. However, we believe that it is beneficial for software to address the more general notion of tensor fields (cf. [7]). A scalar field is then a tensor field of rank 0 and a vector field is a tensor field of rank 1.

Note that both these operations need knowledge about the dimension of the underlying manifold. The gradient operator must know this because, of course, the `Vector Field` has as many components as the dimension of the manifold. But also the Laplacian needs the dimension information. When analyzing the computations of various derivatives, one identifies (in 2D and 3D) four different cases:

1. Cartesian 2D
2. Curvi-linear 2D
3. Cartesian 3D
4. Curvi-linear 3D

In terms of software design, we are aware of two approaches to handling these different cases. The first is when the tensor field itself has operations that carry out these operations. If the coordinate system changes, the tensor field module can be replaced, so that the correct formula is used. This approach is used in Sophus. Note that these formulas depend only on continuous abstractions. This implies that the tensor field abstraction should not deal with discrete approximations. The discrete approximations are carried out by another module, `PD Scalar Field`, where the partial derivatives in the computational coordinates are approximated.

The second approach, which is used in Overture, uses an auxiliary class (`Operators`) to do numerical differentiation. The separation between the representation of tensor fields and the derivative computations is then increased. The separation of concerns between continuous and discrete differentiation is, however, not acknowledged, and we believe that a combination of both approaches would yield a more flexible framework. Besides the variation of the `Operators` module, different curvi-linear mappings may be varied through the use of different `Mappings`, as discussed above.

Sometimes it is desirable to represent a discretization operator as a discrete operator, for instance, if Poisson's equation $\triangle p = f$ is to be discretized as $M\,P = F$, where P and F are discrete representations of p and f, respectively, and the (usually sparse) matrix M is a discrete representation of $\triangle$. Again, we stress that the construction of discrete operators of this kind should be based on coordinate free abstractions. For the Laplacian, we can construct the matrix given a domain:

$$\texttt{createMatrix}_{\triangle}\texttt{: Domain} \rightarrow \texttt{Matrix}.$$

On a curvi-linear domain, the derivatives of the mapping are used to adjust the matrix entries. Our discussions above on variation points for coordinate free abstractions apply in this case as well. Overture supports the construction of discrete differentiation operators in a fashion similar to above. We finally note that our discussions on coordinate free abstractions are not restricted to finite differences. Overture provides operators for finite volumes as well. Finite elements have been used in Sophus [11], and work on finite volume methods is in progress. For these methods, weak formulations of the PDEs is used.

4. Coordinate Free Framework Examples

Previously, we have used trivial model problems to illustrate coordinate free concepts. In this section, we discuss coordinate free numerics for more realistic problems, in connection with existing OO frameworks. In Section 4.1, a seismic simulation carried out with the Sophus framework is presented. This application is described in detail in Haveraaen *et al.* [10]. In Section 4.2, we present simulations of the incompressible Navier-Stokes equations for fluid motion. This is done using the Compose framework which is based on Overture. See Åhlander [14,15] for more details on Compose and this particular application. In Section 4.3, we comment on simi-

larities and differences between the two frameworks.

4.1 Seismic Wave Modeling

Simulation of seismic waves is interesting for instance within the oil industry, where it is used to investigate oil reservoir models. It is also a good example of an application in which the physics often suggests various simplifications. In coordinate free form, we have the following model:

$$\begin{aligned} \rho\frac{\partial^2\vec{u}}{\partial t^2} &= \nabla\cdot\sigma + \vec{f}(t),\\ \sigma &= \Lambda(e),\\ e &= \mathcal{L}_{\vec{u}}(g). \end{aligned} \tag{5}$$

Here, ρ is the density, $\vec{u}$ is the particle displacement, $\vec{f}$ is an external force, σ and e are the stress and strain tensors respectively, Λ is the stiffness tensor of order 4, and g is the geometric tensor of the coordinates. The Lie derivative along a vector field $\vec{u}$ is denoted $\mathcal{L}_{\vec{u}}$. The scalar field density ρ and the stiffness tensor field Λ are given data that vary within the physical domain, modeling the varying geophysical properties of the rocks. A main advantage of the coordinate free form of these equations is that we may use a curvi-linear coordinate system adapted to the folding of geological boundaries. This may improve the numerical accuracy and reduce the amount of computation needed. In the Sophus architecture, the main program is written using the coordinate free interface (see Fig. 6).

```
void advance(
  DiffTensor<ScalarField> &un,       // u at timestep n
  DiffTensor<ScalarField> &unm1,     // u at timestep n-1
  const DiffTensor<ScalarField> &g,  // coordinate system
  const DiffTensor<ScalarField> &Lambda,
  const DiffTensor<ScalarField> &rho,// seismic properties
  const int &stoptime)               // stoptime
{
  DiffTensor<ScalarField> unp1;       // u at timestep n+1
  DiffTensor<ScalarField> e, sigma, a;// intermediate values
  for (int t=0; t < stoptime; t++) {
    e = Lie(un,g);
    sigma = apply(Lambda,e);
    a = div(sigma)/rho + fsource(t);
    unp1 = 2*un-unm1+a*(dt*dt);
    unm1 = un;
    un = unp1;
  }
}
```

Fig. 6. Sample Sophus code. C++ functions corresponding to the operators in Equation (5) form the core of the program.

The main program is then configured using various submodules depending on the model at hand. If, for instance, a simulation is to be carried out around a bore hole with rotational symmetry, the program is configured with a tensor module that computes derivatives accordingly. The configuration may also be used for varying the rock model class. If the rock is considered isotropic, Λ takes a particularly simple form which depends only upon two independent coefficients, to be compared with 21 in the general case. At present, three different classes of rock models are supported in different Sophus modules. This Sophus application uses finite differences on a staggered grid.

The seismic simulations of Eq. (5) show that coordinate free abstractions promote reuse when simulating the same PDE in various situations. We also comment on a situation where the mathematical model is refined even more. The demand for a more precise elastic wave simulation in reservoir zones makes it interesting to study a poro-elastic model:

$$\begin{aligned} \rho_{11}\frac{\partial^2\vec{u}}{\partial t^2} + \rho_{12}\frac{\partial^2\vec{U}}{\partial t^2} &= \nabla\cdot\sigma + \vec{f}(t) + b\left[\frac{\partial\vec{U}}{\partial t} - \frac{\partial\vec{u}}{\partial t}\right],\\ \rho_{12}\frac{\partial^2\vec{u}}{\partial t^2} + \rho_{22}\frac{\partial^2\vec{U}}{\partial t^2} &= \nabla s + \vec{f}(t) - b\left[\frac{\partial\vec{U}}{\partial t} - \frac{\partial\vec{u}}{\partial t}\right],\\ \sigma &= \Lambda(e) + \mathcal{Q}(\epsilon),\\ s &= M(e) + R(\epsilon),\\ e &= \mathcal{L}_{\vec{u}}(g),\ \epsilon = \nabla\cdot\vec{U}. \end{aligned} \tag{6}$$

This model couples the displacement vectors $\vec{u}$ for the solid and $\vec{U}$ for the fluid. The model is described in detail in Haveraaen *et al.* [10]. In this context, we point out that even if the poro-elastic model is considerably more complex, the coordinate free abstractions of the original model (5) can be reused to a high degree. We are just applying the same operators in more complex expressions.

4.2 The Incompressible Navier-Stokes Equations

A common model for many fluid motion simulations is given by the incompressible Navier-Stokes equations. The equations may be expressed in a coordinate free form as follows:

$$\vec{u}_t = -(\vec{u}\cdot\nabla)\vec{u} - \nabla p + \nu\sigma\vec{u} \tag{7}$$

$$\nabla\cdot\vec{u} = 0 \tag{8}$$

Here, $\vec{u}$ is the (2D or 3D) velocity, p is the pressure,

and ν is the viscosity. For numerical computations, we may choose another form of the equations. Following Henshaw and Kreiss [16], we derive an elliptic constraint for p by letting $\nabla \cdot$ act on Eq. (7) and using Eq. (8). We obtain

$$\triangle p = - \nabla \cdot (\vec{u} \cdot \nabla)\vec{u} \tag{9}$$

A numerical simulation based upon a coordinate free formulation may now use Eqs (7) and (9). This example demonstrates some additional points regarding coordinate free PDE software. In Eq. (7), we observe not only the standard coordinate free differentiation operators discussed earlier, but also the operation $(\vec{u} \cdot \nabla)\vec{u}$, a convective derivative. The interface for coordinate free software may vary from application to application, and a coordinate free PDE framework should therefore, ideally, provide good mechanisms for extensibility. Equation (9) provides us with yet another fine point to notice. Even though the right-hand side may be expressed using standard coordinate free building blocks such as the divergence, $\nabla\cdot$, and the Laplacian, $\triangle$, it is difficult for the framework to compute it efficiently. In 2D Cartesian coordinates, for instance, we may use Eq. (8) together with the simple formulas for the differentiation operators to simplify the right-hand side significantly:

$$\triangle p = - u_x^2 - 2v_x u_y - v_y^2. \tag{10}$$

When simulating the incompressible Navier–Stokes equations in the Compose framework, which is based on the coordinate free differentiations on curvi-linear grids provided by Overture[2], we found it pragmatically motivated to develop dedicated equation objects for the Cartesian 2D formulation of the equations (Fig. 7). If desirable, a corresponding development for other coordinate systems (rotational symmetry or 3D, for instance) could be included in the framework. We also emphasize that we still enjoy the coordinate free concepts of the curvi-linear grids, illustrated in Figure 8. The lesson learnt from this example is that even if a coordinate free PDE framework should be based on pure coordinate free operations, it should also be open for compromises as exemplified above. A purely coordinate free approach to solve Eqs (7) and (8) using finite elements is also possible (see Grant *et al.* [11]).

[2] Currently, research has been initiated to develop versions of Compose which use other tools, Cogito, as a supporting layer [1].

```
class INSpressure :  public SpaceEquation
{
  GridFunction u,v; // Components of the solution vector
  ...
  void virtualUpdateSpaceRhs( GridFunction &rhs )
  {
  rhs =                    // compute right hand side
       -u.x()*u.x() - 2.0*v.x()*u.y() - v.y()*v.y();
  }
}
```

Fig. 7. Sample Compose code. The right hand side of Eq. (10) is computed. The formula is optimized for Cartesian 2D and the derivates are computed coordinate free in the sense of curvilinear grids. The gridfunctions `u` and `v` belongs to the class `INSpressure`, which inherits from `SpaceEquation`.

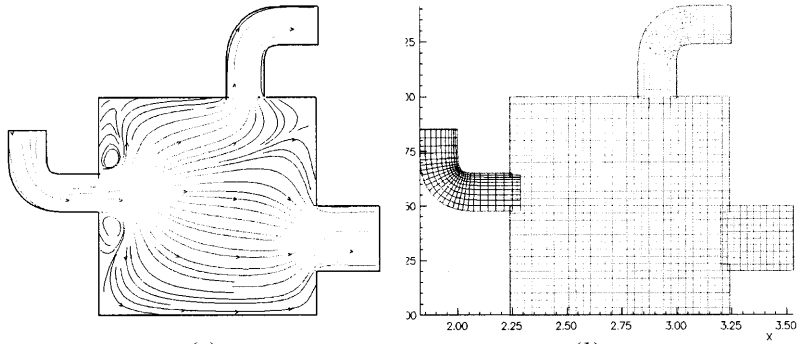
4.3 Comment on Various Framework Architectures

The previous examples were presented mainly to give examples of coordinate free frameworks used in practice. It is also of interest to compare the different framework designs used in Sophus (Section 4.1) and Compose (Section 4.2). Even though both systems are developed in C++, their architectures differ, particularly with respect to the treatment of variation points.

The Sophus framework supports coordinate free mechanisms on the tensor level. Using a technique similar to parametrization using C++ templates, it is possible to configure an application program at link time, to adapt the program to different needs (coordinate systems, parallel vs serial program versions, etc.) The program variations are thus modeled at link time, with the advantage that any run-time overhead is eliminated, and it is possible to use modules which provide only a minimal interface.

Compose, on the other hand, uses Overture's coordinate free mechanisms for curvi-linear, composite, grids. The Compose environment is interactive. It is possible to construct Compose applications where both equations and numerical methods are varied in run-time, using inheritance and dynamic binding. Compose also offers some dedicated debugging facilities [17]. An interactive environment of this kind may be useful when developing and testing new mathematical models.

The choice between compile-time and run-time variation points is not clear-cut, since both approaches have their benefits. In this context it is interesting to observe the possibilities of generic programming, see for instance, Czarnecki and Eisenecker [18]. Using these techniques, it should



(*a*) (*b*)

Fig. 8. A numerically simulated flow in a chamber. Fluid flows into the chamber from the left and out via the two outflows. The geometry is modeled with four curvi-linear grids.

actually be possible to offer both possibilities in the same framework.

5. Summary

Coordinate free abstractions for OO PDE frameworks are abstractions that hide the underlying representation of coordinate systems. By expressing a numerical PDE simulation using coordinate free abstractions, the simulation may transparently use different coordinate systems, which often is called for due to different mathematical or physical models. It is therefore desirable that PDE frameworks provide coordinate free tools. We have identified two different ways to support coordinate free abstractions. First, they may be supported at the 'level of the manifold', embedded in a Cartesian space, and where differentiation operators are implemented using an extrinsic view of the manifold. This is suitable for curvi-linear grids which can be used to construct composite grids. Secondly, coordinate free tools may be based at the 'tensor level', where a pure intrinsic view is taken of the manifold. This is advantageous in order to exploit properties such as rotational symmetry, and it can also be applied to curvi-linear grids.

We have exemplified the use of coordinate free abstractions by two applications; seismic modeling using Sophus and fluid simulations using Compose/Overture. The Sophus architecture allows the same main program to be varied at link time. The seismic application provides variation of different coordinate systems and several other orthogonal variations, serial vs parallel execution, different classes of rock models, etc. The Compose framework enjoys coordinate free abstractions on composite curvi-linear grids using the Overture library, which allows run-time variation using dynamic binding. Both compile-time and run-time variation possibilities have their benefits.

In the context of PDE simulation frameworks, there are several important variation points, such as numerical accuracy, parallel or serial versions etc. Focussing on coordinate free abstractions, some obvious variation points are different coordinate system, different mappings from the computational domain to the physical domain, the dimension of the physical domain, and the very choice of *which* coordinate free differentiation operators should be supported. Finer variation points include whether derivatives of the mappings should be computed in advance and stored in memory or computed on request, and whether they should be numerically approximated or computed analytically (if possible). As mentioned earlier, it is unlikely that one single PDE framework will suit all applications. Different frameworks will continue to provide different points of variation, but we believe variation of coordinates to be a central issue.

Our examples have primarily used finite differences to highlight the advantages of coordinate free PDE abstractions. However, we stress that the underlying principles belong to the continuous abstraction level. Coordinate free abstractions are therefore relevant also when other discretization techniques are used.

Acknowledgement

Research funded via a grant from the Norwegian research council, NFR-project 123585/410.

References

1. Thuné, M., Åhlander, K., Ljungberg, M. *et al.* (2001) Object-oriented modeling of parallel PDE solvers. In:

Boisvert, R., Tang, P. (Editors), The Architecture of Scientific Software, Kluwer Academic, Boston, 159–174
2. Bruaset, A.M., Langtangen, H.P. (1997) A comprehensive set of tools for solving partial differential equations; Diffpack. In: Dæhlen, M., Tveito, A. (Editors), Numerical Methods and Software Tools in Industrial Mathematics, Birkhäuser, Boston, 61–90
3. Brown, D., Henshaw, W., Quinlan, D. (1999) Overture: An object-oriented framework for solving partial differential equations on overlapping grids. In: Henderson, M. E., Anderson, C. R., Lyons, S. L. (Editors), Object-oriented Methods for Interoperable Scientific and Engineering Computing. SIAM, Philadelphia
4. Cummings, J.C., Crotinger, J.A., Haney, S.W. *et al.* (1999) Rapid application development and enhanced code interoperability using the POOMA framework. In: Henderson, M. E., Anderson, C. R., Lyons, S. L. (Editors), Object-oriented Methods for Interoperable Scientific and Engineering Computing. SIAM, Philadelphia
5. Houstis, E.N., Rice, J.R., Weerawarana, S. *et al.* (1998) PELLPACK: A problem-solving environment for PDE-based applications on multicomputer platforms. ACM Transactions on Mathematical Software, 24(1), 30–73
6. Mattsson, M. (1996) Object-oriented frameworks, a survey of methodological issues. Technical Report CODEN:LUTEDX/(TECS-3066)/1-130/(1996), Department of Computer Science, Lund University
7. Åhlander, K., Haveraaen, M., Munthe-Kaas, H. (2001) On the role of mathematical abstractions for scientific computing. In: Boisvert, R., Tang, P. (Editors), The Architecture of Scientific Software, Kluwer Academic, Boston, 145–158
8. Iserles, A., Munthe-Kaas, H., Nørsett, S.P., Zanna, A. (2000) Lie-group methods. Acta Numerica, 9, 215–365
9. Munthe-Kaas, H. (1999) High order Runge–Kutta methods on manifolds. Applied Numerical Mathematics, 29, 115–127
10. Haveraaen, M., Friis, H.A., Johansen, T.A. (1999) Formal software engineering for computational modeling. Nordic Journal of Computing, 6(3), 241–270
11. Grant, P., Haveraaen, M., Webster, M. (2000) Coordinate free programming of computational fluid dynamics problems. Scientific Programming, 8(4), 211–230
12. Schutz, B. (1980) Geometrical Methods of Mathematical Physics. Cambridge University Press
13. Rumbaugh, J., Jacobson, I., Booch, G. (1999) The Unified Modeling Language Reference Manual. Addison Wesley Longman, Reading, MA
14. Åhlander, K. (1999) An extendable PDE solver with reusable components. In: Kudriavtsev, V.V., Kleijn, C.R. (Editors), Computational Technologies for fluid/thermal/structural/chemical systems with industrial applications, volume PVP-Vol. 397–1, 39–46. ASME.
15. Åhlander, K. (1999) An Object-Oriented Framework for PDE Solvers. PhD thesis, Uppsala University, Dept. of Scientific Computing, Uppsala, Sweden
16. Henshaw, W., Kreiss, J.O. (1995) Analysis of a difference approximation for the incompressible Navier-Stokes equations. Technical Report LA-UR-95-3536, Los Alamos National Laboratory, Los Alamos, NM
17. Åhlander, K. (2001) Verifying extensions of OO frameworks: An example from scientific computing. In: Sanchez, B. *et al.* (Editors), Information systems development, vol 2, pp 275–378
18. Czarnecki, K., Eisenecker, U. (2000) Synthesizing objects. Concurrency: Practice and Experience, 12, 1347–1377