Towards an
intensional scientific programming methodology

Joey Paquet  
jpaquet@ift.ulaval.ca

John Plaice  
plaice@acm.org

May 7, 1997

Abstract

Scientific programming is a key element for the advancement of all pure and applied sciences. Unfortunately, the languages currently used for scientific programming are not well suited to the expression of the multidimensional and intensional aspects of differential equations. First, we propose a new "intensional notation" to close the gap between differential notation and intensional programming. Second, we propose an extension of Lucid for the natural expression of differential or tensor equations, and we apply this extension to two examples, plasma simulation and relativistic gravity simulation.

1 Introduction

In all scientific domains, it is now common to do computer simulations. These simulations, which we will collectively designate by the name scientific programming, normally correspond to the operational version of a set of differential equations defining invariants between the variables of a system placed in a multidimensional continuum. These equations, multidimensional and intensional, allow one to describe complex phenomena in a simple and elegant manner.

Currently, the programming of such equations, most of the time in C or in FORTRAN, does not reflect their original simplicity. The fundamental reason is that the semantic constructions used in these languages are inappropriate for the definition of the multidimensional and intensional aspects of differential equations.

The situation gets worse when programming tensor equations: a single tensor equation can imply dozens of ordinary differential equations, and the multidimensional and intensional aspects must be made explicit, thereby exploding the number of equations to be programmed.

The Lucid language [1, 10] manipulates multidimensional dataflows in an intensional manner. Unlike the C and FORTRAN languages that are currently used for scientific programming, Lucid is (1) intensional and multidimensional, (2) declarative, (3) implicitly parallel and (4) based on a demand-driven computation approach. These four characteristics make Lucid an excellent candidate for scientific programming.

In order to prove this affirmation, we shall show the partial translation into Lucid of two programs, one using the differential calculus and the other using the tensor calculus. The first, TRISTAN, written in FORTRAN, simulates plasmas. The second, written in C, simulates the trajectory of particles in a rotating binary black hole system. These two programs are simple, while still being representative of scientific programming.

Certain improvements will be made en route to allow Lucid to correctly express the equations governing the two example problems. We begin by defining an intermediate tensor notation to simplify the translation into intensional programming.
2 Tensor notation

In order to establish a formal transition from tensor notation to that of intensional programming, we have devised a hybrid notation between the two. Tensors, generalizations of vectors, are multilinear objects used to express equations in physics and engineering, independently of the coordinate system or the inertial frame of reference. For example, one of Einstein’s equations for general relativity has the form

\[ G_{ab} = 8\pi T_{ab} \]

thereby linking Einstein’s rank-2 tensor of space-time curvature \( G_{ab} \) with the rank-2 energy-stress tensor \( T_{ab} \) [2].

The indices \( a \) and \( b \) in the equation pertain to each coordinate dimension, for example \((t, x, y, z)\) for Cartesian coordinates. Hence the simple equation above means

\[
\begin{align*}
G_{tt} &= 8\pi T_{tt} & G_{tx} &= 8\pi T_{tx} & G_{ty} &= 8\pi T_{ty} & G_{tz} &= 8\pi T_{tz} \\
G_{xt} &= 8\pi T_{xt} & G_{xx} &= 8\pi T_{xx} & G_{xy} &= 8\pi T_{xy} & G_{xz} &= 8\pi T_{xz} \\
G_{yt} &= 8\pi T_{yt} & G_{yx} &= 8\pi T_{yx} & G_{yy} &= 8\pi T_{yy} & G_{yz} &= 8\pi T_{yz} \\
G_{zt} &= 8\pi T_{zt} & G_{zx} &= 8\pi T_{zx} & G_{zy} &= 8\pi T_{zy} & G_{zz} &= 8\pi T_{zz}
\end{align*}
\]

In spherical coordinates \((t, r, \theta, \phi)\), the same equation means

\[
\begin{align*}
G_{tt} &= 8\pi T_{tt} & G_{tr} &= 8\pi T_{tr} & G_{t\theta} &= 8\pi T_{t\theta} & G_{t\phi} &= 8\pi T_{t\phi} \\
G_{rt} &= 8\pi T_{rt} & G_{rr} &= 8\pi T_{rr} & G_{r\theta} &= 8\pi T_{r\theta} & G_{r\phi} &= 8\pi T_{r\phi} \\
G_{\theta \theta} &= 8\pi T_{\theta \theta} & G_{\phi \phi} &= 8\pi T_{\phi \phi} & G_{\phi \phi} &= 8\pi T_{\phi \phi} & G_{\phi \phi} &= 8\pi T_{\phi \phi}
\end{align*}
\]

The indices to the right of tensors can be placed below, as subscripts, or above, as superscripts. Subscripts normally correspond to contravariant components, while subscripts normally correspond to covariant components. The same notation, with simultaneous use of subscripts and superscripts, can also be used for non-tensors, as is the case for Christoffel symbols [5].

When the same letter simultaneously appears as a subscript and as a superscript, Einstein’s summation convention states that an implicit summation must take place across the entire set of coordinate dimensions. For example, the Ricci rank-2 tensor is defined from the Riemann-Christoffel rank-4 tensor [5]

\[ R_{ab} = R_{a\dot{b}}^{\dot{a}} \]

which implies a contraction over the contravariant superscript and the second covariant subscript, i.e.

\[ R_{ab} = R_{a\dot{b}}^{\dot{a}} + R_{a\dot{x}}^{\dot{x}} + R_{a\dot{y}}^{\dot{y}} + R_{a\dot{z}}^{\dot{z}} \]

or, for spherical coordinates

\[ R_{ab} = R_{a\dot{b}}^{\dot{a}} + R_{a\dot{x}}^{\dot{x}} + R_{a\dot{\theta}}^{\dot{\theta}} + R_{a\dot{\phi}}^{\dot{\phi}} \]

In the two cases, this equation corresponds to 16 different equations where the occurrences of \( a \) and \( b \) are instantiated, as we saw above for the Einstein tensor.

If the coordinate system has \( m \) dimensions, a tensor \( T \) with \( p \) indices has \( m^p \) components. We can consider \( T \) to be an intensional object that varies in a finite set of possible worlds. For example, in a quadridimensional geometry, \( R_{ab} \) varies in 16 worlds and \( R_{abcd} \) in 256 worlds.

However, tensors are also intensional in another sense. Normally, one is interested in a tensor field, in which a tensor is associated with each point in space, and not with a particular tensor. In fact, what is typically of interest is to know the relationship between a tensor’s values at one point and at a neighboring point, infinitely close in differential geometry. We will therefore add to the above tensor notation a left subscript, to designate the value of a tensor, no longer where we are situated, but rather along in the indicated direction. For example \((r+\delta)R_{ab}\) denotes the tensor \( R_{ab} \) at the point \( \delta \) away in the \( r \) direction. Similarly, \((r+\delta)(\phi+\epsilon)R_{ab}\) denotes the tensor \( R_{ab} \) at the point \( \delta \) away in the \( r \) direction and \( \epsilon \) away in the \( \phi \) direction. We suppose here that \((r+\delta)(\phi+\epsilon)R_{ab} = (\phi+\epsilon)(r+\delta)R_{ab} \), which will be the case in all the examples that we examine.
What is important here is that $R$ becomes a doubly intensional object: it not only varies in the finite space of its components, but also in just plain space. Clearly, there is a close relationship between the two, since the space dimensions in which the tensor varies precisely define the possible worlds of its components.

Finally, we allow ourselves to also use left superscripts, in order to enumerate similar vectors or tensors, as in $E_{ab}, E^{ab}, \ldots, E_{ab}$.

To summarize, right subscripts and superscripts are used to designate vector and tensor components. Left subscripts are used to designate displacement in space. Left superscripts are used to designate the terms of an enumeration. The above notation might appear to be complicated, but it allows us to not overload the typical right-subscript position: it will no longer be used to denote several different concepts.

Now that the notation has been defined, the link to Lucid must be defined. The latter treats dimensions in an abstract manner. However, as we have seen above, some dimensions can be considered to be "special". This is the case, for example, for the dimensions defining a geometry, such as $(t, x, y, z)$ and $(t, r, \theta, \phi)$ in the above cases.

The difficulty in Lucid comes from the fact that it above all allows the manipulation of scalar fields, while more complicated fields, such as vector or tensor fields, cannot be expressed in a natural manner. At a given point, the latter vary according to the "special" space dimensions as enumerated above, but do not vary in other dimensions.

We will therefore allow the declaration of certain special dimensions defining "space". The notation

\[
\text{space } x, y, z;
\]
declares a tridimensional space with dimensions $(x, y, z)$, while

\[
\text{space } t, x, \theta, \phi;
\]
declares a quadridimensional space having dimensions $(t, r, \theta, \phi)$.

The declaration of these special dimensions allows a natural tensor declaration. Suppose that space varies in $(t, x, y, z)$. Then the declaration

\[
\text{tensor } R_{-}^{--}, T_{--}^{--};
\]
declares $R$ to be a rank-4 tensor, whose first, third and fourth components are covariant, while the second component is contravariant; and $T$ as being a rank-2 tensor whose components are covariant. In order to use $R$ in an expression, one can either use it in an intensional manner in an expression where all the other tensors have the same form and are without indices; or else with indices, such as in $R_{--1}^{--}j_{k}^{--l}$, which must be declared beforehand; or else explicitly, as in

\[
i, j, k, l : (x, y, z);
\]
This last notation means that the indices $i, j, k$ and $l$ can take as values the dimensions $x$, $y$ and $z$. Vectors are rank-1 tensors, so the following declaration defines two vectors $E$ and $F$:

\[
\text{tensor } E_{--}, F_{--};
\]

Now it is common to want to rotate vectors, and means must be provided for doing so. This can take place by permuting the space dimensions in the index declarations. The following declarations are commonly used:

\[
a : (x, x, y);
\]
\[
b : (x, y, x);
\]
\[
c : (y, z, z);
\]
Hence $E_{b}$ corresponds to the vector $E$ unrotated, $E_{a}$ to the same vector rotated to the right, and $E_{c}$ to the same vector rotated to the left.

3 Example: Differential equations

TRISTAN (Tridimensional Stanford code) is a program written in FORTRAN for the simulation of plasmas. Developed by the late Oscar Buneman [8] at Stanford University, this program has been used to simulate plasmas at many different scales, as in the interaction between the solar wind and the terrestrial magnetosphere [4], solar disturbances [9] and galaxy formation [6].
However, the TRISTAN code is much more complex than the equations upon which it is based. These equations manipulate flows in five dimensions, all simulated using unidimensional arrays. Not every equation requires the explicit manipulation of all the dimensions, but the semantics of the FORTRAN operators does not allow one to abstract out the unreferenced dimensions, which implies an unnecessary duplication of code. Furthermore, many similar operations on permutations of dimensions must take place. The translation into Lucid will allow these equations to be naturally translated in a succinct manner.

Much of the TRISTAN code was already translated in [7, 8]. The two most important results from this translation were that the Lucid code closely resembles the original code and that the inherent parallelism passed from 27 in the FORTRAN version to the number of cells in the corresponding Lucid program, while increasing the level of locality of the computations. However, these translations were made directly from the FORTRAN code rather than from the intensions of the differential equations. We will take a more intensional approach below.

**General behavior**

The core of the TRISTAN code is based on Maxwell’s equations for electromagnetic fields

\[
\begin{align*}
\frac{\partial B}{\partial t} &= -\nabla \times E \\
\frac{\partial D}{\partial t} &= \nabla \times H - j \\
\nabla \cdot D &= \rho \\
\nabla \cdot B &= 0
\end{align*}
\]

and for Lorentz’s equations for particle displacement

\[
\begin{align*}
\frac{d(mv)}{dt} &= q(E + v \times B) \\
\frac{dr}{dt} &= v
\end{align*}
\]

Note that \(D = \varepsilon E\) and that \(B = mH\).

Basically, TRISTAN “runs” Maxwell’s equations in a discrete manner. Space is partitioned into 3-dimensional cells and each cell memorizes a value for each of the components \(zB, yB, zB, zE, yE\) and \(zE\), where \(zF\) means component \(z\) of vector \(F\).

 Charged particles (electrons or positive ions) move within the cells, possibly changing cells, according to the electric and magnetic fields. Since the particles are charged, their movement itself affects the electric field.

 Since differential equations are being manipulated, a leapfrogging technique is used to ensure reasonable accuracy. Therefore, in each cell, the values of the components given above do not all correspond to the values at the origin of the cell, but rather to the following values:

\[
\begin{align*}
 zE_{x}\equiv zE_{(x+.5)y} \\
 yE_{x}\equiv yE_{(y+.5)x} \\
 zE_{y}\equiv zE_{(x+.5)y} \\
 zB_{y}\equiv zB_{(y+.5)(x+.5)} \\
 yE_{y}\equiv yE_{(y+.5)(x+.5)} \\
 zB_{y}\equiv zB_{(x+.5)(y+.5)}
\end{align*}
\]

This scheme is given in Figure 1. To simplify the presentations, we use our intensional syntax, by writing:

\[
\begin{align*}
 E' &\equiv E_{(y+.5)} \\
 B' &\equiv B_{(x+.5)(y+.5)} \quad \text{with } \delta = .5
\end{align*}
\]
Here, we are only showing the aspects that change, with index $b$ corresponding to the component that is of interest, $a$ corresponding to the component rotated to the left and $c$ corresponding to the component rotated to the right.

Here, we will examine in detail the two aspects of the program that we consider to be fundamental:
1. computing the curl in order to update the magnetic and electric fields;
2. computing the particle displacement.

Computing the curl

Is it through the derivation of the curl for Lucid that we show how one can pass from physics equations to a LUCID program. We begin with the definition for the curl:

$$\nabla \times \mathbf{F} = \hat{x} \left( \frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z} \right) + \hat{y} \left( \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x} \right) + \hat{z} \left( \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right)$$

If we retain the intensional syntax for components, the curl then becomes:

$$\nabla \times \mathbf{F} = \left( \epsilon F_{(c+\delta)} - \epsilon F_{(c-\delta)} \right) - \left( \epsilon F_{(a+\delta)} - \epsilon F_{(a-\delta)} \right)$$

where $\delta$ is an infinitely small value. But this same expression can also be used to define finite difference equations, by allowing the $\delta$ to become a standard real value.

Now we are interested in the computation of the following expressions:

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E}$$
$$\frac{\partial \mathbf{D}}{\partial t} = \nabla \times \mathbf{H} - \mathbf{j}$$
Since we will have to discretize space, we are in fact going to be interested in the values of

\[
E' \equiv E_{(b+\delta)} \\
B' \equiv B_{(a+\delta)(c+\delta)}
\]

which gives us:

\[
(\nabla \times B)_{(b+\delta)} = \left( a B_{(b+\delta)(c+\delta)} - a B_{(b+\delta)(c-\delta)} \right) - \left( c B_{(a+\delta)(b+\delta)} - c B_{(a+\delta)(b-\delta)} \right)
\]

\[
= \left( a B' - a B'_{(c-\delta)} \right) - \left( c B' - c B'_{(a-\delta)} \right)
\]

and

\[
(\nabla \times E)_{(a+\delta)(c+\delta)} = \left( a E_{(a+\delta)(c+\delta)} - a E_{(a+\delta)(c-\delta)} \right) - \left( c E_{(a+\delta)(b+\delta)} - c E_{(a+\delta)(b-\delta)} \right)
\]

\[
= \left( a E' - a E'_{(c+\delta)} \right) - \left( c E'_{(a+\delta)} - c E' \right)
\]

If we consider \( \delta = .5 \), then we can now define the equivalent in Lucid.

Our implementation will use five dimensions: three for space (x, y and z), one for time (dt) and one for the enumeration of particles that are found within cells (dl). Here is the definition for dimensions and indices:

- space x,y,z;
- dimension dt,dl;
- a : (x,x,y);
- b : (x,y,z);
- c : (y,z,x);

We repeat our equations (with \( \delta = .5 \)):

\[
\nabla \times B' = \left( a B' - a B'_{(c-1)} \right) - \left( c B' - c B'_{(a-1)} \right)
\]

\[
\nabla \times E' = \left( a E' - a E'_{(c+1)} \right) - \left( c E'_{(a+1)} - c E' \right)
\]

Here are the Lucid equations:

\[
dprev.d(F) = F - prev.d(F);
\]

\[
dnext.d(F) = next.d(F) - F;
\]

\[
diff.i.j(f,F) = f.j(F-i);
\]

\[
curl(f,F) = diff.a.c(f,F) - diff.c.a(f,F);
\]

In other words:

\[
curl(dnext,E') = diff.c,b(dnext,E') - diff.b,c(dnext,E');
\]

\[
curl(dprev,B') = diff.c,b(dprev,B') - diff.b,c(dprev,B');
\]

This formal derivation gives the same result as in [7, 8], but it has the advantage of making the result evident, in addition to not relying on the FORTRAN program.

**Particle displacement**

The position of each particle at time \( t \) is altered by three parameters: (1) the particle's velocity, (2) the particle's acceleration due to the local electric field on its charge and (3) the rotational acceleration due to the local magnetic field on its displacement.

To compute the changes in position and velocity of a particle, the values for the forces at the particle's exact location must be interpolated:

\[
E'' = \frac{E'_{b-1} + E'_{b}}{2}
\]

\[
B'' = \frac{B'_{a-1} + B'_{a}}{2}, \quad B = \frac{B'_{c-1} + B'_{c}}{2}
\]
This is equivalent to
\[ E' = \text{avgprev.b(E')} \]
\[ B' = \text{avgprev.a(avgprev.c(B'))} \]
\[ \text{avgprev.d(F)} = 0.5 * (F + \text{prev.d(F)}) \]

Elements \( E'' \) and \( B'' \) are the values of \( E \) and \( B \) interpolated to the origin of each of the cells. Forces (2) and (3) in the above paragraph are computed through interpolation, using the general formula
\[ F_{b+3} = F + \delta \times (F_{b+1} - F) \]

which is directly translated into
\[
\text{interpolate.x(X,interpolate.y(X,interpolate.z(X,F)))}
\]
where
\[
\text{interpolate.d(X,F) = F + frac(X_d) * dnext.d(F)};
\]
end;

where \( X \) is the 3-dimensional vector containing the particle's coordinates.

The resulting velocity is computed using the Lorentz equation
\[
\frac{d(mv)}{dt} = q(E + v \times B)
\]
interpolated in time and to which a relativistic \( \gamma \) factor for mass increase has been added [6]
\[
\frac{(v_{t+1} - v_t)}{\delta t} = \frac{q}{\gamma m_0} \left( E + \left( \frac{v_{t+1} + v_t}{2} \right) \times B \right)
\]

where \( q \) is the charge, \( m_0 \) is the particle's rest mass and \( E \) and \( B \) are the electric and magnetic fields at the particle's initial position.

This equation must be transformed into a computable form, which gives the following system of equations [6]:
\[
\begin{align*}
x_{t+1} &= x_t + u_{t+1}; \\
u_{t+1} &= \gamma(u_t)u_t \\
u_1 &= u_0 + \left( \frac{2}{1 + B_0^2} \right) (u_0 + u_0 \times B_0) \times B_0 + E_0 \\
u_0 &= \eta(u_0)u_0 - E_0 \\
B_0 &= \gamma(u_0)q \frac{B_0}{2m_0} \\
E_0 &= \frac{q - E_0}{2m_0} \\
\gamma(v) &= \frac{c}{\sqrt{c^2 + v^2}} \\
\eta(v) &= \frac{c}{\sqrt{c^2 - v^2}}
\end{align*}
\]

This system is now directly programmable in Lucid, once again using the leapfrogging technique.

\[
\text{move(X,U,B,E) = (Xnew,Unew)}
\]
where
\[
\begin{align*}
X_{\text{new}} &= X + U_{\text{new}}; \\
U_{\text{new}} &= g(U1) \times U1; \\
U1 &= U0 + \text{cross}(f \times (U0 + \text{cross}(U0,B0)),B0) + E0; \\
U0 &= \text{gneg}(U) \times U + E0; \\
f &= 2.0 / (1.0 + \text{sqnorm}(B0)); \\
B0 &= g(U0) \times (q / (2.0*\text{m}^2)) \times E; \\
E0 &= (q / (2.0*\text{m}^2)) \times E;
\end{align*}
\]
\[ g(V) = \frac{c}{\sqrt{\text{square}(c) + \text{sqnrm}(V)}}, \]
\[ \text{g neg}(V) = \frac{c}{\sqrt{\text{square}(c) - \text{sqnrm}(V)}}, \]
\[ \text{sqnrm}(F) = \text{square}(F_a) + \text{square}(F_b) + \text{square}(F_c), \]
\[ \text{cross}(F1,F2) = (\text{square}(c) - \text{square}(a) - \text{square}(c) - \text{square}(c)), \]
\[ \text{square}(F) = F^2; \]
end;

Summary

The example given in this section shows that it is quite possible to take an existing FORTRAN program, to study and analyze it and then to produce an equivalent Lucid program that is much much shorter and clearer and in which parallelism is much greater. We have also shown that our intensional tensor syntax simplifies the creation of Lucid programs starting from differential equations.

4 Example: Tensor equations

The tensor example that is presented below corresponds to the displacement of a particle in the neighbourhood of a black hole, as predicted by some interpretations of general relativity [2]. This program is sufficiently complex to explain fundamental problems, but can be summarized in a few lines. The presentation below is just an outline.

To define the problem, suppose the Boyer-Linquist coordinate system \((t, r, \theta, \phi)\). The black hole has mass \(M\), angular momentum \(J\) and rotates in direction \(\phi\). The fundamental tensor \(g_{ij}\), defining the geometry's metric at that point, has form

\[
\begin{pmatrix}
g_{tt} & 0 & 0 & g_{t\theta} \\
0 & g_{rr} & 0 & 0 \\
0 & 0 & g_{\theta\theta} & 0 \\
g_{t\theta} & 0 & 0 & g_{\phi\phi}
\end{pmatrix}
\]

with, at each point \((t, r, \theta, \phi)\)

\[ g_{tt} = -(1 - \frac{2M}{r}) \]
\[ g_{t\theta} = -\left(\frac{2aMr}{r} \sin^2(\theta) \right) \]
\[ g_{\phi\phi} = g_{t\theta} \]
\[ g_{rr} = \frac{\Sigma}{\Delta} \]
\[ g_{\theta\theta} = \Sigma \]
\[ g_{\phi\phi} = r^2 + a^2 + 2Mra^2 \sin^2(\theta) \sin^2(\theta) \]
\[ a = \frac{J}{M} \]
\[ \Delta = r^2 - 2Mr + a^2 \]
\[ \Sigma = r^2 + a^2 \cos^2(\theta) \]

In Lucid, we begin with the definition of space and of the indices associated with the space dimensions:

space i,j,k,l : t,r,theta,phi;
Function kerr_metric defines the metric at a given point

kerr_metric(p"k) = g_i_j
where
\[ g_{t\theta} = -(1 - \frac{2M}{r}/\Sigma) \]
\[ g_{\phi\phi} = -\left(\frac{2aM}{r} \sin(\theta) / \Sigma \right) \sin(\theta) / \Sigma \]
\[ g_{r\theta} = g_{\phi\phi} \]
\[ g_{rr} = \frac{\Sigma}{\Sigma} \]
\[ g_{\theta\theta} = \Sigma \]
\[ g_{\phi\phi} = r^2 + a^2 + 2Mra^2 \sin^2(\theta) \sin^2(\theta) / \Sigma \]
\[ a = \frac{J}{M} \]
\[ \Delta = r^2 - 2Mr + a^2 \]
\[ \Sigma = r^2 + a^2 \cos^2(\theta) \]
\[ g_{\theta,\theta} = \Sigma ; \]
\[ g_{\phi,\phi} = r^2 + a^2 + 2M\Sigma r^a a^* \sin(\theta) \sin(\theta) \]
\[ \times \sin(\theta)\Sigma \sin(\theta)/\Sigma \Sigma ; \]
\[ g = 0 ; \]
\[ a = J/M ; \]
\[ \Delta = r^2 - 2M\Sigma + a^2 ; \]
\[ \Sigma = r + a^2 \cos(\theta) \cos(\theta) ; \]
\[ r = p^2/\theta ; \]
\[ \theta = p^2/\theta ; \]
end;

where the last \( g = 0 \) declares that the default value is 0.

The problem of interest is to compute the acceleration \( a^i \) undergone by a particle moving with velocity \( u^i \) at point \( p^i \):

\[
a^i = -\left( \Gamma^i_{jk} u^j u^k \right)
\]
\[
\Gamma^i_{jk} = \frac{g^{ik} \left( g_{jk, i} + g_{kj, j} - g_{ik, j} \right)}{2}
\]

where

\[ g_{ij, k} = \frac{\partial g_{ij}}{\partial x^k} \]

In Lucid, we currently write \( g_{ij, k} \) as \( g_{i, j, k} \), although this decision may change if tensor derivatives are introduced.

\[
\text{ KerrGeodesic}(p^i, u^i) = a^i
\]

where

\[ a^i = -(\text{Gamma}^i_{j, k} u^j u^k) ; \]
\[ \text{Gamma}^i_{j, k} = g^{i-1} \left( (g_{j, k} + g_{k, j} - g_{i, j}) / 2 \right) ; \]
\[ g^{i-1} = \text{inverse}(g_{i, j}) ; \]
\[ g_{j, k} = \text{kerrMetric}(p^{1+(k\times\text{delta})}) - g_{j, j} ; // \text{a revoir} \]
\[ g_{i, j} = \text{kerrMetric}(p^{1}) ; \]
end;

We now have an idea of how to translate tensor equations into our version of Lucid.

5 Conclusion

The study of several examples from scientific programming has shown that the differential and tensor calculuses are naturally intensional concepts. It is precisely the inability of conventional languages to express this intensionality that makes scientific programming complex, even if the original program is not complex. The use of an intensional language such as Lucid greatly simplifies programming complexity.

What is still missing is an efficient implementation. Current work involves automatically generating GLU programs from (Tensor) Lucid programs. We claim that the extra space dimensions can be used to assist in the automatic partitioning of such programs to create appropriate GLU grains. Should this approach work, it could even be refined to use some kind of geodesic to define the partitions.

The ultimate goal of this project is to allow efficient execution of programs that correspond very closely to the equations in the original domain of study.

References


