Diderot: A Parallel DSL for Computing on Multi-Dimensional Tensor Fields

http://diderot-language.cs.uchicago.edu

Nicholas Seltzer† Lamont Samuels John Reppy Gordon L. Kindlmann

Department of Computer Science; University of Chicago

ABSTRACT
We describe ongoing work developing Diderot, a new parallel domain-specific language (DSL) for computing on continuous fields reconstructed from multi-dimensional images. Diderot uses a syntax similar to mathematical conventions for expressing convolution, differentiation, and tensor operations, making it easy to code computationally complex algorithms. In addition to being easy to code, programs written in Diderot will be compiled for several different parallel architectures, with many optimizations handled automatically by the compiler.

Keywords: Image Data Visualization, Image Analysis, Parallelism, Domain-Specific Language

1 INTRODUCTION
Many visualization programs are difficult to write, even though the algorithms may involve simple mathematical principles. We frequently need to take derivatives of multi-dimensional data or handle tensors which result from such operations. Such ideas can usually be expressed with mathematical notation in one short line, but implementing them may require hundreds of lines of code or require the use of complex libraries. Diderot is a parallel DSL with high-level mathematical notation for expressing computations with tensors and tensor fields. Diderot naturally handles fields of any dimension and tensors of any order. Moreover, Diderot programs will be portable to multiple hardware targets without changing the code at all. Our goal is a language for easily managing computations on continuous fields reconstructed from discrete datasets that leads to efficient, optimized, portable, and parallel code.

Diderot is inspired by previous parallel DSLs. One such language is Scout, a general data-parallel language [6] compiled for GPUs. Another DSL, Shadie, is a GPU-based volume visualization framework which emphasizes code simplicity [7]. Other work seeks to compile a particular domain of algorithms to a range of parallel architectures such as SMPs, clusters, and GPUs. For example, Liszt is a DSL for expressing computations on finite element meshes and for compiling code to a variety of architectures [3]. Diderot follows this approach, restricting the domain of the language to computations on continuous tensor fields (including scalar and vector fields).

2 LANGUAGE ABSTRACTIONS
Diderot has several built-in types and operations for expressing high-level mathematical computations and abstracting away discrete image data. The syntax of Diderot expressions mirrors conventional mathematical notation, making programs easier to read.

The tensor is the basic computational type. Tensors are declared as: tensor[d1, d2, ..., dm]. The [d1, d2, ..., dm] (di > 1) is the shape of an mth order tensor. Common tensor types have aliases, e.g., real means tensor[] and vec3 means tensor[3]. Diderot supports addition (+) and subtraction (−) of tensors, as well as common vector and tensor products (i.e., inner p • q, cross p × q, outer p ⊗ q).

†nseltzser@cs.uchicago.edu

The image type represents multi-dimensional arrays of tensors, and stores meta-data about how the array is oriented in world space. The image can have any number of dimensions and store tensors of any order. Images are declared as: image(n) [d1, d2, ..., dm]. n is the dimension of the image array (n > 0) and [d1, d2, ..., dm] is the shape of the tensors stored in the image. For example, to load a 3D image of scalars:

image(3)[i] my_data = load("data.nrrd");

Diderot uses univariate piecewise polynomial kernels to reconstruct continuous fields from discrete data by separable convolution. Each kernel has a Ck continuity level that limits the number of times a field reconstructed with the kernel may be differentiated to k. Diderot provides kernels such as C3 Cubic B-spline bspln3, C1 Catmull-Rom ctmr, and C0 tent for linear interpolation.

The field type combines kernels and discrete data to create the abstraction of smooth tensor fields over continuous world-space. Fields are declared as field#k(n) [d1, d2, ..., dm], where k is the continuity level. Convolving 3D image my_data with a cubic B-spline creates C3 scalar field my_field:

field#2(3)[i] my_field = bspln3 @ my_data;

Besides convolution, fields are created by adding, subtracting, and scaling existing fields, as well as by differentiation:

field#1(3)[3] gradient = Vmy_field;
field#0(3)[3,3] Hessian = V⊙Vmy_field;

V, V×, and V⊙ are higher-level functions that apply analytic differentiation to fields. Fields are not pre-computed; they are abstract functions that can be evaluated at any location to produce a tensor. Kernel differentiation and convolution are not computed until a field is probed at a particular world-space location.

3 COMPOSITION MODEL
Diderot is a C-like language extended with high-level mathematical notation for expressing computations on tensors and tensor fields. Diderot organizes the computation into mostly-autonomous strands, which are the unit of parallelism. Each strand has local state and an update method. The execution model is bulk-synchronous and deterministic. Each iteration updates every strand independently. Execution continues until all strands either stabilize, producing output, or die, with no output.

A Diderot program has a simple structure composed of three parts: global declarations, strand definitions, and strand initialization. Global declarations include variables readable by all strands. The strand definition includes some parameters, strand-local variables (and their initialization), and an update method. Finally, strand initialization creates all the strands with the correct parameters. Like shader languages, Diderot global variables are immutable, but local strand-instance variables are mutable. The following is a simple Diderot program which creates 50 strands that output the numbers 0.0 to 49.0:

int numStrands = 50;
strand my_strand (int i) {
output real out = real(i);
update (stabilize);
}
initially { my_strand(i) treatment: i in 0..(numStrands - 1) };

The image type requires the use of complex libraries. Diderot is a parallel DSL with bulk-synchronous parallel domain-specific language (DSL) for computing on continuous tensor fields (including scalar and vector fields).
Not only does Diderot make code easier to write, it enables efficient executables for a variety of architectures, taking advantage of the highly parallel nature of such algorithms. The Diderot compiler currently supports both sequential and SMP executables, and a GPU (OpenCL) version is under development. Traditionally, parallel programs must be tuned for different platforms by hand, but, because Diderot is designed for a specific domain of computation, it is possible for the compiler to generate efficient code for significantly different target platforms. Domain knowledge also allows the compiler to take advantage of target features such as SSE vector instructions automatically. Because Diderot is implemented as a compiler, instead of as a library, it is able to optimize across the whole program. For example, it can eliminate redundancies in probing a field’s value, gradient, and Hessian.

4 Example Applications

The language abstractions and computational model discussed above support a wide range of visualization and analysis algorithms. Direct volume rendering is one natural application of Diderot. Direct volume rendering creates an image by casting one ray per pixel through a field and determining color from local field properties [4]. The computation of each ray is independent, so we create one strand per ray, with an update function that increments along the ray and updates the color according to the transfer function. Local curvature, summarized in a geometry tensor, G, can be used in transfer functions [5] (Figure 1):

\[
\text{vec3 step} = h \cdot \text{normalize}(\nabla F(p));
\]

\[
\text{if} (|\nabla F(p)| > \text{threshold \&\&} -|\nabla F(p+\text{step})| + 2 \cdot |\nabla F(p) - |\nabla F(p+\text{step})| = 0)
\]

A more refined technique would create a particle system, initialized with particles evenly spread over the image domain. The solution would be computed with one strand per particle, using Newton-Raphson iterations to move them along image gradients towards maxima of gradient magnitude.

![Figure 3: Edge detection of a face.](image)

Figure 3: Edge detection of a face.

5 Future Work

While we have been able to write several non-trivial Diderot programs, Diderot is still a work in progress. Work is being done to allow strands to communicate and share information. We also plan to support the dynamic creation of strands after computations have started and to add a global computation step to be executed after each iteration. These additions will increase the range of algorithms supported by Diderot. We are also working to make Diderot code callable as a C library. With callable libraries, it will be easy to combine Diderot computations with other processing, and to chain Diderot computations together. Future work will include porting Diderot to MPI clusters. We expect one of the first applications of Diderot will be in education, allowing students to focus on high-level algorithmic design and still get highly parallel executables that efficiently process real-world datasets.

References

1. Introduction

Motivation: Efficient analysis and visualization of multi-dimensional fields

A Parallel Domain Specific Language

Diderot: A Parallel Domain Specific Language for Computing on Multi-Dimensional Fields

Program structure

1. Introduction
2. Diderot Details
3. Language Details
4. Future Work
5. Usability Improvements

Diderot follows this approach, being restricted to computations on continuous tensor fields. The image is to be used to reconstruct a continuous field. The notation makes it easy for domain experts to rapidly implement a variety of image analysis and visualization algorithms, the simplicity of Diderot's notation makes it an ideal high-level mathematical abstraction language for multi-dimensional fields.

The source code in figure 2 shows each section of the program structure. An example of the use of Diderot's mathematical abstractions is included in Figure 3. A tensor field was designed to demonstrate computations of curvature using second derivatives of the field. The final image was colored with a transfer function involving tensor[3,3] functions. The output is then compared to the final image and the result is shown.

The field was stabilized, in which case their state contributes to the output. This is a typical example of a possible application for Diderot.

The final image was colored with a transfer function involving tensor[3,3] functions. The output is then compared to the final image and the result is shown.

$\kappa = \text{trace}(G)$

$G = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

$\nabla \kappa = \frac{\partial}{\partial x} \kappa$