Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis

Gordon Kindlmann, Charisee Chiw, Nicholas Seltzer, Lamont Samuels, John Reppy

THE UNIVERSITY OF CHICAGO
Computation Institute
Scientific Context & Motivation

- Scientists need software to show and measure structure in large complex image datasets
- Creating new visualization/analysis tools is an essential part of the scientific process
Programmers want **portable** parallel languages.

Scientists need to **rapidly** implement variety of new programs.

**Goal:** speed the development of portable parallel methods of 3D scientific visualization and analysis.

Creating vis/analysis tools is hard to do.

Increasing range of:
- Imaging **modalities**
- Imaging **applications**
- Vis & analysis **algorithms**

Increasing **data size** → Need **parallel** computing → Rapidly shifting parallel computing architectures.
Triangle of language strengths (courtesy Pat Hanrahan)

- **Performance**
  - C
  - C++

- **Generality**
  - Javascript
  - Python
  - Ruby
  - Lua

- **Productivity**
  - Domain Specific Languages

- “Why not write a library?”

- DSL advantages:
  1. Code can be concise, idiomatic (types, syntax, operations)
  2. Compiler analysis, optimizations
  3. Express parallel execution apart from OS, hardware (CPU/GPU)

- Expert C/C++ coders like libraries

Goal: Open up Sci Vis research to a larger user community
Related DSL research

- **Vivaldi** [Choi-VIS-2014]: Volume rendering, processing in Python-like DSL on distributed GPU clusters

- **ViSlang** [Rautek-VIS-2014]: Slangs (procedural, declarative, functional) interactively combined

- **Scout** [McCormick-VIS-2004] [McCormick-JPC-2007] [Jablin-IPDPS-2011] [McCormick-WOLFHPC-2014]: compile data- or task-parallel programs on grids, using LLVM toolchain

- Other DSLs discussed in paper
- Diderot’s strength: **idiomatic mathematical abstractions**
Diderot computes on fields, not samples

Convolve image data (a) with kernel to get continuous field (c)

\[ F = \text{ctmr} \ast \text{image("hand.nrrd")}; \]

Field\#1(2)[] \( F = \text{ctmr} \ast \text{image("hand.nrrd")}; \)

Field\#N(D)[S]: Continuous field: \( \mathbb{R}^D \rightarrow \text{tensors shape S} \)

[]: scalar, [3]: 3-vector, [3,3]: 3x3 matrix (Appendix A gives grammar)
Example complete program: isocontour sampling

```alloy
field#1(2)[] F = c4hexic ⊗ image("hand.nrrd");
input int size0; input int size1;
input int stepsMax = 10;
input real epsilon = 0.0001;
input vec2 dir0; input vec2 dir1;
input vec2 orig;
strand isofind(vec2 pos0) {
  output vec2 pos = pos0;
  int steps = 0;
  update {
    // Stop after too many steps or leaving field
    if (steps > stepsMax || !inside(pos, F))
      die;
    // one Newton-Raphson iteration
    vec2 delta = -normalize(∇F(pos)) * F(pos)/|∇F(pos)|;
    pos += delta;
    if (|delta| < epsilon)
      stabilize;
    steps += 1;
  }
}
initially { isofind(orig + ui*dir0 + vi*dir1) |
  vi in 0..(size1-1), ui in 0..(size0-1) }
```

Globals are immutable; used for program inputs

Strands are bulk synchronous

Strand state, including output

`update` method implements algorithm

Initialization of collection of strands with comprehension notation

Legible math!
Volume rendering soft isosurfaces

```plaintext
field#0(1)[3] cmap = tent ◦ image("isobow.nrrd");
field#4(3)[] V = bspln5 ◦ image("canny.nrrd");
field#4(3)[] F = V - isoval;
...
function real alpha(real v, real g) = max(0, 1 - |v|/(g*thick));
...
strand raycast(int ui, int vi) {
  real transp = 1;
  vec3 rgb = [0,0,0]; output vec4 rgba = [0,0,0,0];
  update {
    if (rayN > camVspFar) { stabilize; }
    real val = F(x);
    vec3 grad = -∇F(x);
    real a = alpha(val, |grad|);
    real shade = max(0, normalize(grad)•light);
    rgb += transp*a*(0.2 + 0.8*shade)*color(x);
    transp *= 1 - a;
  }
  stabilize {
    real a = 1-transp;
    if (a > 0) rgba = [rgb[0]/a, rgb[1]/a, rgb[2]/a, a];
  }
}
initially [ raycast(ui, vi)
  | vi in 0..iresV-1, ui in 0..iresU-1 ];
```

Isosurface is zero level-set

[Levoy-CGnA-1988]

Over operator with pre-multiplied alphas

set final output rgba
Volume rendering material boundaries

• How to show material boundaries?

• Canny edge [Canny-PAMI-1986]:
  • $|\nabla v|$ maximal w.r.t motion along $\nabla v/|\nabla v|$
  • $\nabla |\nabla v| \cdot \nabla v/|\nabla v| = 0$

• Change one line of Diderot code:
  • `field#4(3)[] F = V - isoval;`
  • `field#2(3)[] F = \nabla |\nabla V| \cdot \nabla V/|\nabla V|;`

• For shading, Diderot computes $\nabla F$
  • involves 3rd derivatives (!)
Canny edges in real CT scan

• There is no isosurface that captures the bone surface
• Canny edge surface shows underlying value (novel vis)

Data: Callum Ross, University of Chicago
Rendering flow field structure

Data: Resampling by Tino Weinkauf of Navier-Stokes simulation by S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo

\[ \text{field}#4(3)[3] \, V = \text{bspln5} \odot \text{image("flow.nrrd")}; \]
\[ \text{field}#3(3)[3] \, F = \left( \frac{V}{|V|} \right) \cdot \left( \nabla \times V \right) \left/ \left| \nabla \times V \right| \right); \]

- Normalized Helicity [Degani-AIAAJ-1990]
Rendering anisotropy of diffusion tensor field

\[
\text{field}(3)[3,3] \ V = \text{bspln5 } \circ \ \text{image("dti.nrrd")}; \\
\text{field}(3)[3,3] \ E = V - \text{trace}(V) * \text{identity}[3]/3; \\
\text{field}(3)[] \ F = \sqrt{3.0/2.0} * |E|/|V| - \text{isoval};
\]

Compare with original definition [Basser-JMRB-1996]

\[
D = D - \langle D \rangle I.
\]

\[
\text{FA} = \sqrt{3 \frac{\sqrt{D:D}}{2}}.
\]

Not just for volume rendering!

Data: Centre for Functional MRI of the Brain, John Radcliffe Hospital, Oxford University
Streamlines in flow field

vec2{} x0s = load("seeds.txt"); // list of seedpoints
real h = 0.02;
int stepNum = 200;
field#1(2)[2] V = bspln3 ⊗ image("flow.nrrd");
real arrow = 0.1; // scale from |V(x)| to arrow size
strand sline(vec2 x0) {
    int step = 0;
    vec2 x = x0;
    output vec2{} p = {x0}; // start streamline at seed
    update {
        if (inside(x, V)) {
            x += h*V(x + 0.5*h*V(x)); // Midpoint method
            p = p @ x; // append new point to streamline
        }
        step += 1;
        if (step == stepNum) {
            // finish streamline with triangular arrow head
            vec2 a = arrow*V(x); // length of arrow head
            vec2 b = 0.4*[−a[1],a[0]]; // perpendicular to a
            p = p@(x−b); p = p@(x+a); p = p@(x+b); p = p@x;
            stabilize;
        }
    }
}
initially [ sline(i, x0s{i}) | i in 0..length(x0s)-1 ];
**Compilation**

- Compiler written in SML/NJ
- Three stages of intermediate representation (IR)
  - “EIN” IR is like lambda calculus meets Einstein summation notation
- Produces identities:
  - $\nabla \cdot (\nabla \times V) = 0$
  - $\text{Trace}(u \otimes v) = u \cdot v$
- Section 5.1 of paper
- Use **clang** to compile executable or C library
Compile to executable or C Library

• Stand-alone executable w/ command-line interface
  • each input has corresponding option
    • input real isoval = 10; ⇒ ... -isoval 10 ...

• Compile to library, with API for
  • Setting inputs, retrieving outputs
    • ISO_InVarSet_isoval(ISO_World_t *wrld, float v);
    • ISO_OutputGet_pos(ISO_World_t *wrld, Nrrd *data);
  • Initializing, stepping through computation

• Appendix B: 2D particle system example
• Let’s watch 3D particle system go ...
(snapshots from interactive demo shown during talk)
Speedup curves (on CPU)

- Significant improvement in speedup relative to previous 2012 paper in Programming Language Design and Implementation (PLDI)
Performance numbers

<table>
<thead>
<tr>
<th>Program</th>
<th>Teem</th>
<th>Seq.</th>
<th>1P</th>
<th>6P</th>
<th>12P</th>
<th>16P</th>
<th>Seq.</th>
<th>1P</th>
<th>6P</th>
<th>12P</th>
<th>16P</th>
<th>OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>vr-lite</td>
<td>19.93</td>
<td>8.63</td>
<td>9.51</td>
<td>2.57</td>
<td>2.94</td>
<td>3.20</td>
<td>7.46</td>
<td>7.52</td>
<td>1.36</td>
<td>0.74</td>
<td>0.59</td>
<td>1.43</td>
</tr>
<tr>
<td>illust-vr</td>
<td>86.16</td>
<td>44.30</td>
<td>48.55</td>
<td>8.65</td>
<td>5.61</td>
<td>5.19</td>
<td>38.12</td>
<td>38.28</td>
<td>7.00</td>
<td>3.79</td>
<td>2.88</td>
<td>4.32</td>
</tr>
<tr>
<td>lic2d</td>
<td>3.03</td>
<td>1.59</td>
<td>1.64</td>
<td>0.33</td>
<td>0.19</td>
<td>0.16</td>
<td>1.56</td>
<td>1.51</td>
<td>0.28</td>
<td>0.15</td>
<td>0.12</td>
<td>1.09</td>
</tr>
<tr>
<td>ridge3d</td>
<td>7.92</td>
<td>5.96</td>
<td>6.36</td>
<td>1.12</td>
<td>0.62</td>
<td>0.56</td>
<td>5.22</td>
<td>5.26</td>
<td>0.93</td>
<td>0.50</td>
<td>0.39</td>
<td>1.77</td>
</tr>
</tbody>
</table>

Execution times in seconds, averaged over 10 runs

• “Teem” = hand-coded C, not parallel (no pthreads)
• Intel Xeon E5-2687W (16 cores), Ubuntu 12.04.
• OpenCL w/ NVIDIA Tesla K20c, using NVIDIA’s CUDA 6.0 driver
• Appendix C compares with hand-written OpenCL
Ongoing Work

• Stronger math abstractions
  • Declarative mathematical statement of algorithm
  • Time-varying fields (time as special dimension)

• Better computing
  • New backends: CUDA and MPI (for larger datasets)
  • Better GPU performance through OpenCL
  • New fields: (higher-order) Finite Element Meshes

• Better usability: debugger, GUI generation
Conclusions

• Good progress on an ambitious goal

• Diderot good for:
  • Writing legible vis programs that run in parallel
  • Trying new sci vis methods in terms of fields, tensors

• Diderot not (yet) good for:
  • Working directly on grids (e.g. Marching Cubes, level-set segmentation, per-pixel classification)
  • Fast execution on big data essential, rather than fast implementation
Works cited


Thank you

• Anonymous reviewers for constructive comments
• National Science Foundation CCF-1446412

• Data: University of Utah SCI group, NIH NIGMS grant P41GM103545 | Callum Ross, University of Chicago | Resampling by Tino Weinkauf of Navier-Stokes simulation by S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo | Xavier Tricoche, Purdue University | Centre for Functional MRI of the Brain, John Radcliffe Hospital, Oxford University | Wolfgang Kollmann, UC Davis

• Reproducibility! (Even before a release...)

• Example programs from this talk will be here: https://github.com/Diderot-Language/examples

• Google Group: https://goo.gl/kXpxhV

• Questions?