Diderot: a Domain-Specific Language for Portable Parallel Scientific Visualization and Image Analysis  [Chiw-PLDI-2012] [Kindlmann-VIS-2015]

Please interrupt me with questions!

Gordon Kindlmann
GLK@uchicago.edu
20 Jan 2016 Geilo Winter School

joint work with: Charisee Chiw, Nicholas Seltzer, Lamont Samuels, Prof. John Reppy
Warning: Diderot is not like Paraview

Paraview

Diderot
• 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
Creating vis/analysis tools is hard to do

Increasing range of:

- Imaging modalities
- Imaging applications
- Vis & analysis algorithms

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

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

Programmers want **portable** parallel languages

Increasing data size → Need **parallel** computing → Rapidly shifting parallel computing architectures
Torrent of new data from microscopy

Digital Light Sheet Microscopy [https://en.wikipedia.org/wiki/Light_sheet_fluorescence_microscopy]

• Compared to confocal microscopy:
  • Fewer photons to get same image quality
  • Less phototoxicity, photobleaching
  • More data: \(~5 \text{ gigabytes / minute}\)
  $\implies \sim7 \text{ terabytes / day}$
Where do pioneer neurons come from; where do they go; how do they find their way?

Pioneer neurons are first to traverse path of nerve

Facial branchiomotor neuron (FBMN) of zebrafish, pioneer migration starts ~16 hpf
Example Data

Prof. Victoria Prince
(Dept. Organismal Biology and Anatomy, University of Chicago), Anastasia Beiriger

• Where do pioneer neurons come from; where do they go; how do they find their way?
• Pioneer neurons are first to traverse path of nerve
• Facial branchiomotor neuron (FBMN) of zebrafish, pioneer migration starts ~16 hpf
Example visualization method [Kindlmann-VIS-2003]

\[
\nabla n^T = -\nabla \left( \frac{g^T}{|g|} \right) = - \left( \nabla g^T - \frac{g \nabla g^T |g|}{|g|^2} \right) \\
= - \frac{1}{|g|} \left( H - \frac{g \nabla (g^T g)^{1/2}}{|g|} \right) = - \frac{1}{|g|} \left( H - \frac{g \nabla (g^T g)}{2 |g|^2 (g^T g)^{1/2}} \right) \\
= - \frac{1}{|g|} \left( H - \frac{g (2g^T H)}{2 |g|^2} \right) = - \frac{1}{|g|} \left( I - \frac{g g^T}{|g|^2} \right) H \\
= - \frac{1}{|g|} (I - nn^T) H.
\]
The C code to implement that
**OpenCL code (for GPUs)**

```c
float4 computeGradient(image3d_t sampler, float4 gradPos, const float gradOffset)
{
    // central differences gradient
    read_image(samplex, linearSampler, (float4)(gradPos.x+gradOffset, gradPos.y, gradPos.z, 0.f));
    read_image(samplex, linearSampler, (float4)(gradPos.x-gradOffset, gradPos.y, gradPos.z, 0.f));
    read_image(samplex, linearSampler, (float4)(gradPos.x, gradPos.y+gradOffset, gradPos.z, 0.f));
    read_image(samplex, linearSampler, (float4)(gradPos.x, gradPos.y-gradOffset, gradPos.z, 0.f));
    read_image(samplex, linearSampler, (float4)(gradPos.x, gradPos.y, gradPos.z+gradOffset, 0.f));
    read_image(samplex, linearSampler, (float4)(gradPos.x, gradPos.y, gradPos.z-gradOffset, 0.f));
    gradient6 = computeGradient(gradOffset);
    gradient5 = computeGradient(gradPos-(float4)(0.f,0.f,gradOffset,0.f), sampler);
    gradient4 = computeGradient(gradPos-(float4)(0.f,gradOffset,0.f,0.f), sampler);
    gradient3 = computeGradient(gradPos-(float4)(gradOffset,0.f,0.f,0.f), sampler);
    gradient2 = computeGradient(gradPos+(float4)(gradOffset,0.f,0.f,0.f), sampler);
    gradient1 = computeGradient(gradPos+(float4)(0.f,0.f,0.f,0.f), sampler);
    return (float4)(gradient6, gradient5, gradient4, gradient3, gradient2, gradient1);
}

float4 computeCurvature(image3d_t sampler, float4 gradPos, const float gradOffset)
{
    float4 gradient6 = computeGradient(gradOffset);
    float4 gradient5 = computeGradient(gradPos-(float4)(0.f,0.f,gradOffset,0.f), sampler);
    float4 gradient4 = computeGradient(gradPos-(float4)(0.f,gradOffset,0.f,0.f), sampler);
    float4 gradient3 = computeGradient(gradPos-(float4)(gradOffset,0.f,0.f,0.f), sampler);
    float4 gradient2 = computeGradient(gradPos+(float4)(gradOffset,0.f,0.f,0.f), sampler);
    float4 gradient1 = computeGradient(gradPos+(float4)(0.f,0.f,0.f,0.f), sampler);
    float4 gradient = computeGradient((gradPos.x, gradPos.y, gradPos.z-gradOffset, 0.f)).x,
            read_imagef(sampler, linearSampler, (float4)(gradPos.x, gradPos.y, gradPos.z+gradOffset, 0.f)).x-read_imagef(sampler, linearSampler, (float4)
            gradPos.x, gradPos.y+gradOffset, gradPos.z, 0.f)).x-read_imagef(sampler, linearSampler, (float4)
            gradPos.x-gradOffset, gradPos.y, gradPos.z, 0.f)).x,
            read_imagef(sampler, linearSampler, (float4)(gradPos.x+gradOffset, gradPos.y, gradPos.z, 0.f)).x-read_imagef(sampler, linearSampler, (float4)
            gradPos.x+gradOffset, gradPos.y+gradOffset, gradPos.z, 0.f)).x)
            return (float4)(k1,k2);
}
```
Time to think about new languages?

From Abelson & Sussman & Sussman *Structure and Interpretation of Computer Programs* (1985):

“First, we want to establish the idea that a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology. Thus, **programs must be written for people to read, and only incidentally for machines to execute.**”
Time to think about new languages?

From Donald Knuth *Literate Programming* (1992):

“Let us change our traditional attitude to the construction of programs: instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.”
Triangle of language strengths (courtesy Pat Hanrahan)

- **Performance**
  - C
  - C++
  - Javascript
  - Python
  - Ruby
  - Lua

- **Generality**
  - Domain Specific Languages

- **Productivity**

  “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**.
What is Diderot best at?

Algorithms with large number of (mostly) independent computations based on local properties of continuous fields:

- Direct Volume Rendering
- Streamlines, Fiber Tractography
- Particle Systems for Image Feature Sampling

i.e.: an important part of traditional “sci vis” research
Matlab, Numpy great for discrete images.

Continuous fields ≠ discrete images

\[
\text{image} = f[x] = g[x] = |\nabla f[x]| \\
|\nabla(f[x] \ast k(x))| \\
g[x] \ast k(x)
\]

(f[x])

(non-linear function of f[x])
Objects versus images

Measurements of objects produce images

Algebraic Vis: “data”

“representation”

underlying object

measured image

Goal of scientific visualization & analysis is to make statements about the underlying objects being studied
Objects versus images

Grid orientation/spacing is property of image (“representation”)

Marching Squares (on grid)  Newton’s method (on field)

Diderot’s support for continuous fields helps get away from grid details towards object properties
Objects versus images

Previous work from 1928:

*La trahison des images*, Magritte

![Painting of a pipe with the text "Ceci n’est pas une pipe." ](http://edc13.education.ed.ac.uk/phild/files/2013/01/magritt-this-is-not-a-pipe.jpg)
Minimal Diderot program example

Square roots of numbers 1..1000 by Heron’s method

// Global definitions
input int N = 1000;
input real eps = 0.000001;

// Strand definition
strand sqroot(real val) {
  output real root = val;
  update {
    root = (root + val/root)/2.0;
    if (|root^2 - val|/val < eps) {
      stabilize;
    }
  }
}

// Strand initialization
initially [ sqroot(real(i)) | i in 1..N ]
Bulk synchronous execution model
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
- Uses `clang` to compile executable or C library
Defining fields from images in Diderot

• Convolve image data (a) with kernel to get continuous field (c)
  \[ \text{field}^{#1(2)}[\cdot] \ F = \text{ctmr} \ ⊗ \ \text{image("hand.nrrd")}; \]
  
• field\#N(D)[S]: C^N continuous field: \( \mathbb{R}^D \to \text{tensors shape } S \)
  
  • []: scalar, [3]: 3-vector, [3,3]: 3x3 matrix (\text{Appendix A} gives grammar)
Revisiting curvature measurement

Direct (coordinate-free) notation encourages basis-independent (representation-independent) code

1. Measure the first partial derivatives comprising the gradient $g$. Compute $n = -g/|g|$, and $P = I - nn^T$.
2. Measure the second partial derivatives comprising the Hessian $H$ (Equation 1). Compute $G = -PHP/|g|$.
3. Compute the trace $T$ and Frobenius norm $F$ of $G$. Then,

$$
\kappa_1 = \frac{T + \sqrt{2F^2 - T^2}}{2}, \quad \kappa_2 = \frac{T - \sqrt{2F^2 - T^2}}{2}.
$$

```
// volume dataset
field#2(3)[] F = bspln3®image("quads.nrrd");
// RGB colormap of K1, K2
field#0(2)[3] RGB = tent®image("rgb.nrrd");
...
update {
  ...
  vec3 g = VF(pos);
  vec3 n = -g/|g|; // or -normalize(g);
  tensor[3,3] H = V®VF(pos);
  tensor[3,3] G = -(P®H®P)/|grad|;
  real disc = sqrt(2*|G|^2 - trace(G)^2);
  real k1 = (trace(G) + disc)/2;
  real k2 = (trace(G) - disc)/2;
  // find material RGBA
  vec3 matRGB = RGB([k1, k2]);
  ...
```
```
"
Volume rendering soft isosurfaces

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]:

$$|\nabla v| \text{ maximal w.r.t motion along } \nabla v / |\nabla v|$$

$$\Rightarrow \nabla |\nabla v| \cdot \nabla v / |\nabla v| = 0$$

Change one line of Diderot code:

```diderot
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)
Rendering flow field structure

\[
\text{field}#4(3)[3] \ V = \text{bspln5} \ ⊗ \text{image("flow.nrrd")};
\]

\[
\text{field}#3(3)[] \ F = (V/|V|) \cdot (\nabla \times V/|\nabla \times V|);
\]

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

\[
\text{field}^{4}(3)[3,3] \ V = \text{bspln5} \odot \text{image}(\text{"dti.nrrd"});
\]

\[
\text{field}^{4}(3)[3,3] \ E = V - \text{trace}(V) \ast \text{identity}[3]/3;
\]

\[
\text{field}^{4}(3)[\ ] \ FA = \sqrt{3.0/2.0} \ast |E|/|V| - \text{isoval};
\]

Compare with original definition [Basser-JMRB-1996]

\[
\text{``} \quad D = D - \langle D \rangle I
\]

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

Not just for volume rendering!
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 ];

Output is set of sequence of points

Legible integration
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 [Chiw-PLDI-2012]
## Performance numbers

<table>
<thead>
<tr>
<th>Program</th>
<th>Teem</th>
<th>Seq.</th>
<th>Diderot (PLDI ’12)</th>
<th>Diderot (this paper)</th>
<th>OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1P</td>
<td>6P</td>
<td>12P</td>
</tr>
<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>
</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>
</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>
</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>
</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


Thank you!

National Science Foundation CCF-1446412

Data: Mouse paw: University of Utah SCI group, NIH NIGMS grant P41GM103545 | Capuchin Skull: Callum Ross, University of Chicago | Vortex flow field: Resampling by Tino Weinkauf of Navier-Stokes simulation by S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo | Double-point stress field: Xavier Tricoche, Purdue University | Diffusion Tensor Brain: Centre for Functional MRI of the Brain, John Radcliffe Hospital, Oxford University | 2D flow field: Wolfgang Kollmann, UC Davis

• Example programs are accumulating here: https://github.com/Diderot-Language/examples
• Google Group: https://goo.gl/kXpxhV
• If want to follow-along with demos, git pull https://github.com/Diderot-Language/examples
(break)
Example running Heron

From a checkout of
https://github.com/Diderot-Language/examples

cd heron

.. '/../vis12/bin/diderotc --exec heron.diderot

./heron --help  (to see usage info)

./heron  (run program)

(on one line)

unu crop -i vrie.nrrd -min 0 0 -max 1 M |
unu jhisto -b 300 300 |
unu quantize -b 8 -o tmp.png

(view tmp.png)
1D Convolution defined

**discrete** data samples $V[]$, **continuous** reconstruction kernel $k$

$$f(x) = (g \ast k)(x) = \int g(t)k(x - t)dt$$

$$= \int \sum_j V[j] \delta(t - j)k(x - t)dt$$

$$= \sum_j V[j] \int \delta(t - j)k(x - t);$$

$$f(x) = \sum_j V[j]k(x - j)$$
2D Convolution examples 1

box, nearest neighbor

“tent” = linear interpolation
2D Convolution examples 2

“ctmr” = Interpolating cubic
“Catmull-Rom” spline

“bspln3” = (non-interpolating)
Cubic B-spline
Reconstruction with nice kernels

Diderot used for Algebraic Vis paper [Kindlmann & Scheidegger 2014]

Reconstruction with simple ("ctmr", top) vs ("c4hexic" bottom) kernel; c4hexic can exactly reconstruct cubics

Fig. 2: Our Invariance Principle illustrated with taxi pick-ups and drop-offs (a), two different samples from a population (b), volume renderings of sampled 3D cubic polynomial (c), and vector glyphs in a 2D flow field (d). The upper pair of adjacent visualizations are of exactly the same underlying data or object, but give different impressions due to arbitrary differences in representation, sometimes beyond the control of the designer. The bottom row demonstrates the Invariance Principle with visualizations that do not depend on representation choice.
2D image sampler/viewer again in a checkout of https://github.com/Diderot-Language/examples

cd vimg
(peruse source code)  (sscand = southern Scandanavia)
ln -s ../data/sscand.nrrd img.nrrd
../../../vis12/bin/diderotc --exec vimg.diderot
./vimg --help (to see usage info)
./vimg -cent 300 400 -fov 30
unu quantize -b 8 -i gray.nrrd -o tmp.png (view tmp.png)
(try again with different kernels; have to recompile each time)
(try again with -w 1 to see gradient magnitude)
./vimg -cent 300 400 -fov 30 -w 2 -iso 1070 -th 40

How to get even thickness line?
Taylor’s Theorem

\[ f(x + \varepsilon) \approx f(x) + f'(x)\varepsilon \]

\[ f(x + \varepsilon) \approx f(x) + \nabla f(x) \cdot \varepsilon \]

And from this can derive Newton’s method...

Compare to Levoy’s paper [Levoy-CGnA-1988]
Example complete program: isocontour sampling

\[
\text{field#1(2)[] } F = \text{c4hexic } \odot \text{ image("hand.nrrd")};
\]

\[
\text{input int size0; input int size1;}
\]

\[
\text{input int stepsMax = 10;}
\]

\[
\text{input real epsilon = 0.0001;}
\]

\[
\text{input vec2 dir0; input vec2 dir1;}
\]

\[
\text{input vec2 orig;}
\]

\[
\text{strand isofind(vec2 pos0) } \{\\
\quad \text{output vec2 pos = pos0;}
\quad \text{int steps = 0;}
\quad \text{update } \{\\
\quad \quad \text{// Stop after too many steps or leaving field }\\
\quad \quad \text{if (steps > stepsMax || !inside(pos, F))}\\
\quad \quad \quad \text{die;}\\
\quad \quad \text{// one Newton-Raphson iteration }\\
\quad \quad \text{vec2 delta = -normalize(∇F(pos)) } \ast F(pos)/|∇F(pos)|;\\
\quad \quad \text{pos += delta;}\\
\quad \quad \text{if (|delta| < epsilon)}\\
\quad \quad \quad \text{stabilize;}\\
\quad \quad \text{steps += 1;}\\
\quad \}\}
\]

\[
\text{initially } \{ \text{isofind(orig + ui*dir0 + vi*dir1) |}\\
\quad \text{vi in 0..(size1-1), ui in 0..(size0-1) } \};
\]
Data: Mouse paw: University of Utah SCI group, NIH NIGMS grant P41GM103545 | Capuchin Skull: Callum Ross, University of Chicago | Vortex flow field: Resampling by Tino Weinkauf of Navier-Stokes simulation by S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo | Double-point stress field: Xavier Tricoche, Purdue University | Diffusion Tensor Brain: Centre for Functional MRI of the Brain, John Radcliffe Hospital, Oxford University | 2D flow field: Wolfgang Kollmann, UC Davis

• Example programs are accumulating here: https://github.com/Diderot-Language/examples
• Google Group: https://goo.gl/kXpxhV

• Please share your thoughts on how to write another paper about Diderot! GLK@uchicago.edu