Discovering Stable Features in Scientific Images (and a bit about Diderot)

Gordon Kindlmann

http://people.cs.uchicago.edu/~glk/
glk@uchicago.edu

Context

- Goal: geometric models of meaningful features for quantitative analysis of large image datasets
  - “Feature” = proxy for structure under scientific study
- Want: generality WRT feature co-dimension
- Method: particle systems for feature sampling
- But: how do we know if some mathematical feature is plausible as an anatomic feature?
Outline

- Features
- Particles
- Scale-space
- Stability with respect to scale
- Results
- Discussion

By “Image” I mean a continuous field

Specimen position arbitrary; want fine structure

\[ g[x] = |\nabla f[x]| \]

\[ f[x] \ast k(x) \]

\[ |\nabla (f[x] \ast k(x))| \]

\[ g[x] \ast k(x) \]
What I mean by mathematical “feature”

- In a continuous & differentiable field \( f(x) \) created by convolution:
  - Feature = positions \( x \) satisfying feature equation
    - e.g. Isocontours: \( f(x) = f_0 \)
    - e.g. Laplacian zero-crossing edges: \( \nabla^2 f = 0 \)
  - (No models, priors, blend of data & smoothing terms..)
  - \((g = \nabla f ; H = \nabla \otimes \nabla f ; H = \sum_i \lambda_i e_i \otimes e_i ; \lambda_1 \geq \lambda_2 \geq \lambda_3)\)
- Combinations of maxima and minima WRT \( e_i \)
  - Ridge Surface (Eberly ’96): \( g \cdot e_3 = 0; \lambda_3 < 0 \)
  - Ridge Line: \( g \cdot e_3 = g \cdot e_2 = 0; \lambda_3 < 0; \lambda_2 < 0 \)
  - Iterative update scheme (Newton optimization) to move closer to feature if near it: we can sample features

Dynamic Particle Systems

- Set of points subject to (particle-image) feature constraint and (interparticle) energy minimization
  \[
  \arg\min_{x_i,N} E = \arg\min_{x_i,N} \sum_{i,j=1}^{N} E_{ij}
  \]
- Hard constraint of particles to feature; no energy

\[
E_{ij} = \Phi \left( \frac{|x_i - x_j|}{\sigma_r} \right)
\]

(demo)
Visual survey of feature types

- Illustrated with hand from Visible Human, Female CT
- Feature types
  - Isosurface
  - Laplacian zero-crossing
  - Ridges & Valleys (“creases”)
    - surfaces or lines
- **Same** particles, many features
  - each little glyph = one particle
  - show local feature ingredients
- Can sample features, but
  - Issues: **scale**, **meaningfulness**

Isosurface

- AKA isocline, isophote, isocontour, level set
- \( f(x) = v_0 \)
Laplacian 0-crossing

- Classical definition of edge
- $\nabla^2 f(x) = 0$; strength $= |\nabla f(x)|$

Ridge Surface

- Maximal surface wrt Hessian minor eigenvector $e_3$
- $\nabla f(x) \cdot e_3(x) = 0$; strength $= -\lambda_3$
Ridge Line

- Maximal curve wrt Hessian minor, medium eigenvectors
- $\nabla f(x) \cdot e_3(x) = 0$, $\nabla f(x) \cdot e_2(x) = 0$; strength = $-\lambda_2$

Valley Surface

- Minimal surface wrt Hessian major eigenvector $e_1$
- $\nabla f(x) \cdot e_1(x) = 0$; strength = $\lambda_1$
Valley Line

- Minimal curve wrt Hessian major, medium eigenvectors
- $\nabla f(x) \cdot e_1(x) = 0$, $\nabla f(x) \cdot e_2(x) = 0$; strength $= \lambda_2$

Scale Space

- Image + continuous family of blurrings
  - feature scale not always known a priori
- Practical requirements
  - Probe at arbitrary points in scale space
  - Efficiently handle real-world 3D datasets
- Scale interpolation based on Lindeberg’s “Gaussian” (soln. to heat eq. in discrete domain)

\[
\frac{\partial L}{\partial s} = s (L[\cdot; s^2] * [1 \ -2 \ 1])
\]
Sampling scale-space feature
Sampling scale-space feature

Particles sampling ridge across scale

Stability?
Sampling scale-space feature

**Localyze in scale**
with particle-Image energy as function of feature strength

Hermite spline interpolation across scale

---

Linear blending across scale not good for scale localization

---

Features | Particles | Scale-space | Scale stability | Results | Discussion
The purpose is not images, it is feature sampling

Feature localization and sampling in space and scale

Glyphs displaying scale

Applications
Applications

Helicity conservation by flow across scales in reconnecting vortex links and knots

Martin W. Scheeler, Dustin Kleckner, Davide Proemi, Gordon L. Kindlmann, and William T.M. Irvine

Department of Physics, Illinois, USA.

PNAS 2014 (to appear)

Application: APS MicroCT (Keith Cheng, PI)

volume rendering courtesy Darin Clark, PSU
Extraction of micro-anatomy

• Extraction of individual photoreceptors from microCT
  • With scale-space particles
  • Lots of structure to be discovered at other scales!
But how do you know if a feature is meaningful?

• A math feature (edge, ridge) may exist in image, but does it represent a physical or real structure?
• Basic idea: it's real if it doesn't move when you blur a little bit (stable with respect to scale)
• Precedent:
  • SIFT: Scale-invariant Feature Transform
  • Marr-Hildreth edge detection

Testing stability with respect to scale

Synthetic data for testing

Z slice

X projection

Isosurface
Stability of ridge lines

Stability of valley surfaces
Discussion

- New imaging modalities, contrast mechanisms, and their combination => proliferation of possibly useful features; this may help show the way
- “Finding” features in 2 senses: where, and which
- From image samples to feature samples
- Future work:
  - Points into polyline trees, polygonal meshes
  - Shape modeling and statistics
Creating new tools is hard

Increasing range of:

- Imaging modalities
- Imaging applications
- Vis & analysis questions → algorithms

Want to **rapidly implement** variety of programs (the “inner loops”)

Diderot helps rapidly develop portably parallel methods of image visualization and analysis

One example problem....

Want **portably** parallel implementations

Increasing **data size** → Need **parallel** computing → Rapidly shifting parallel computing architectures

(switching gears)
Digital Light Sheet Microscopy

• PI: Kevin White, Institute for Genomics and Systems Biology, U of Chicago

• Drosophila embryo-genesis imaged over 20 hours

• Movie shows 2 projections of 3D data

Data: 2 channels x 1700x1700 (XY) x 500 (Z) x ~500 (time) x ~100 genes

Data processing by Luke Peeler, BS/MS

• New Zeiss Z-1 (UChicago KCBD)
• What is the spatial/temporal structure of gene expression, at the resolution of single cells?
• Fancy device, but almost no code!
Diderot  http://diderot-language.cs.uchicago.edu

• Domain-Specific Language for portably parallel analysis and visualization of continuous fields (scalar/vector/tensor)

• Gain **programmer efficiency** and **parallel performance** at cost of algorithmic **generality**

• **Portably** parallel: compiles to multi-core CPUs (pthreads), GPUs (OpenCL)

• High-level notation supports rapid development and mathematically legible code (“from whiteboard to executable”)


What is Diderot best at?

• Algorithms with large number of (mostly) independent computations with local properties of continuous fields, e.g.:
  • Direct Volume Rendering
  • Streamlines, Fiber Tractography
  • Particle Systems for Image Feature Sampling
**Objects versus images** (again)

- Measurements of objects produce **images**

- Goal of scientific vis & analysis is to make statements about the underlying **objects**

---

**Objects versus images** (again)

- Grid orientation/spacing is property of **image**

- Continuous fields (in Diderot) help get away from grid details towards object properties
**Diderot program structure**

- Computation decomposed into collection of mostly autonomous *strands*
- Each strand has state and an `update` method
- `update` implements one iteration of algorithm
  - strands can stabilize, die, new
- Abstractions:
  - Fields: convolution & differentiation of discrete data
  - Parallel computation (CPU vs GPU)
  - Strand communication

**Minimal example**

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

```plaintext
// 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 ];
```
Execution model

Example: sampling isosurfaces

```plaintext
input real isoval = 0.4;
field#1(2)[] F = ctmr@image("ddro.nrrd") - isoval;
int grid = 150;
int stepsMax = 10;
real epsilon = 0.000001;
strand FindZero(vec2 x0) {
    output vec2 x = x0;
    int steps = 0;
    update {  
        if (!inside(x, F) || steps > stepsMax)
            die; // Stop outside domain or after many steps
        if (|∇F(x)| == 0)
            die; // Can’t proceed with zero derivative
        // the Newton-Raphson step
        vec2 dx = normalize(∇F(x)) * F(x)/|∇F(x)|;
        x -= dx;
        if (|dx| < epsilon)
            stabilize; // Converged when step small enough
        steps += 1;
    }
}
initially { FindZero([lerp(0, 1, -0.5, ui, grid-0.5),
                        lerp(0, 1, -0.5, vi, grid-0.5)])
                  | vi in 0..(grid-1), ui in 0..(grid-1) };```

Diagram courtesy J. Reppy
Compilation

- Compiler written in SML/NJ
- Three stages of intermediate representation

- Use `cc` to create executable (with command-line interface) or C library (with API)

Mandelbrot set

```c
// Global definitions
input int reso = 2000;
input real escape = 4.0;
input int maxiter = 1000;
input vec2 center = [0, 0];
input real fov = 2;
field[2][3] cmap = tent @ image("colormap.nrrd");

// Strand definition
strand mandel(vec2 c) {
  vec2 z = c;
  int iter = 0;
  output vec3 rgb = [0, 0, 0];
  update {
    // $z = z^2 + c$
    z = [z[0]^2 - z[1]^2, 2*z[0]*z[1]] + c;
    if (|z| > escape) {
      // point escaped; color based on iter and |z|
      real time = iter - log2(log(|z|)/log(escape));
      rgb = cmap(fmod(log(time), 1));
      stabilize;
    }
  iter += 1;
  if (iter > maxiter) {
    rgb = [0, 0, 0];
    stabilize;
  }
}

// Strand initialization
initially [ mandel([lerp(center[0]-fov, center[0]+fov,
    1, realIdx, reso),
    lerp(center[1]-fov, center[1]+fov,
    1, compIdx, reso)])
  | compIdx in 1..reso, realIdx in 1..reso ];
```
Blood vessel sampling w/ particles

Particles repel each other w/ potential function (strand communication by Lamont Samuels)

Finding valley lines

```
strand valleyline(vec3 initpos) {
    output vec3 x = initpos;
    update {
        vec3 ev = evecs(∇∇F(x));
        vec3 dir = normalize((ev{3}⊙ev{3} + ev{2}⊙ev{2})•∇F(x));

        real fdd = ∇F(x)•dir;
        real sdd = dir•∇∇F(x)•dir;
        vec3 delta = dir*fdd/sdd; // Newton Optimization
        if (|delta| < epsilon) {
            stabilize;
        }
        x -= delta;
    }
}
```

Lung airways (chest CT)
Example visualization method

• Curvature-based transfer functions (Vis’03)

\[
\nabla n^T = -\nabla \left( \frac{g^T}{|g|} \right) = \left( \frac{\nabla g^T}{|g|} - \frac{g\nabla^T|g|}{|g|^2} \right) = \frac{1}{|g|} \left( H - \frac{g\nabla^Tg|g|^2}{2g^Tg/|g|^2} \right) = \frac{1}{|g|} \left( H - \frac{2g^T(Hg)}{2|g|^2} \right) = \frac{1}{|g|} \left( I - \frac{gg^T}{|g|^2} \right) H = \frac{1}{|g|} (1-\mathbf{n}\mathbf{n}^T) H.
\]

The C code to implement that
OpenCL code (for GPUs)

```c
float2 computeCurvature(image3d_t sampler, float4 gradPos, const float gradOffset)
{
    float2 k1 = (t + sqrt(2.f*f*f-t*t))/2.f;
    float k2 = (t - sqrt(2.f*f*f-t*t))/2.f;
    return (float2)(k1,k2);
}
```

Example: curvature measurement

```c
vec3 grad = -VF(pos);
vec3 norm = normalize(grad);
// begin curvature computation
tensor[3,3] H = V*VF(pos);
tensor[3,3] P = identity[3] - norm@norm;
tensor[3,3] G = (P*HP)/lgrad;
real disc = max(0.0, sqrt(2.0*grad^2) - trace(G)^2));
real k1 = (trace(G) + disc)/2.0;
real k2 = (trace(G) - disc)/2.0;
// find material RGBA
vec3 matRGB = RGB([clamp(-1.0, 1.0, 6.0*k1),
                        clamp(-1.0, 1.0, 6.0*k2)]);
```

Direct (coordinate-free) notation encourages and basis-independent code (eventually, dimension-independent code)
Much to do (still) ...

• Mathematically idiomatic field definitions
  • \textbf{lifting}: \text{field}\#1(3)[3] V = \text{ctmr} \odot \text{image(vec.nrrd)} \rightarrow \text{field}\#1(3)[] M = |V| \quad (\text{Charisee Chiw})
  • \textbf{composition}: V = W \circ F, and then differentiation by chain rule
• Fields from point clouds, Finite Element Meshes

• Many possibilities for GUI / IDE:
  • Sliders for all \textbf{input} variables
  • Nicer compiler error messages
  • Stepping debugger
• More parallel targets: MPI, CUDA
• Virtual memory for big datasets

Summary

• Harder and harder to extract knowledge from scientific imaging
• Biologists are most in need of practical parallel tools, and least empowered to create them for their work
• Diderot is (ambitious) work-in-progress
• Is open-source (and undocumented)
  • http://diderot-language.cs.uchicago.edu
• Thanks for your attention! glk@uchicago.edu