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

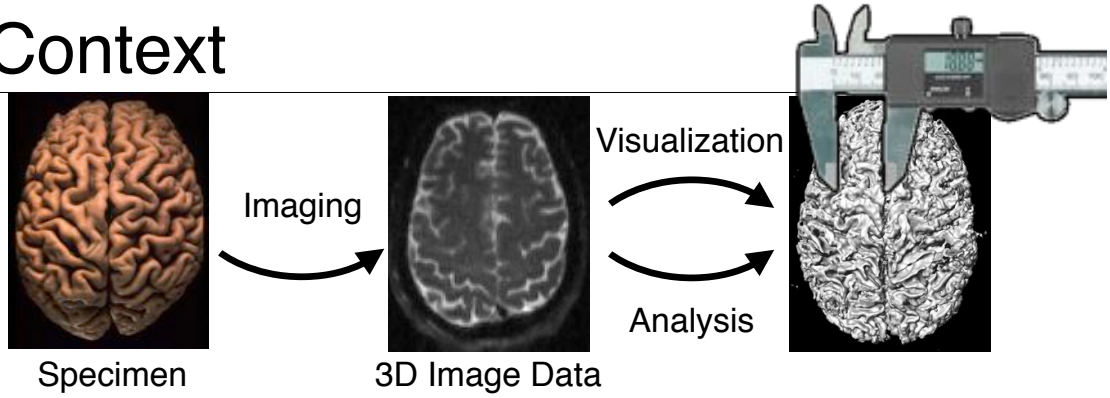
Gordon Kindlmann

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

THE UNIVERSITY OF CHICAGO COMPUTER SCIENCE

Computation Institute

Context



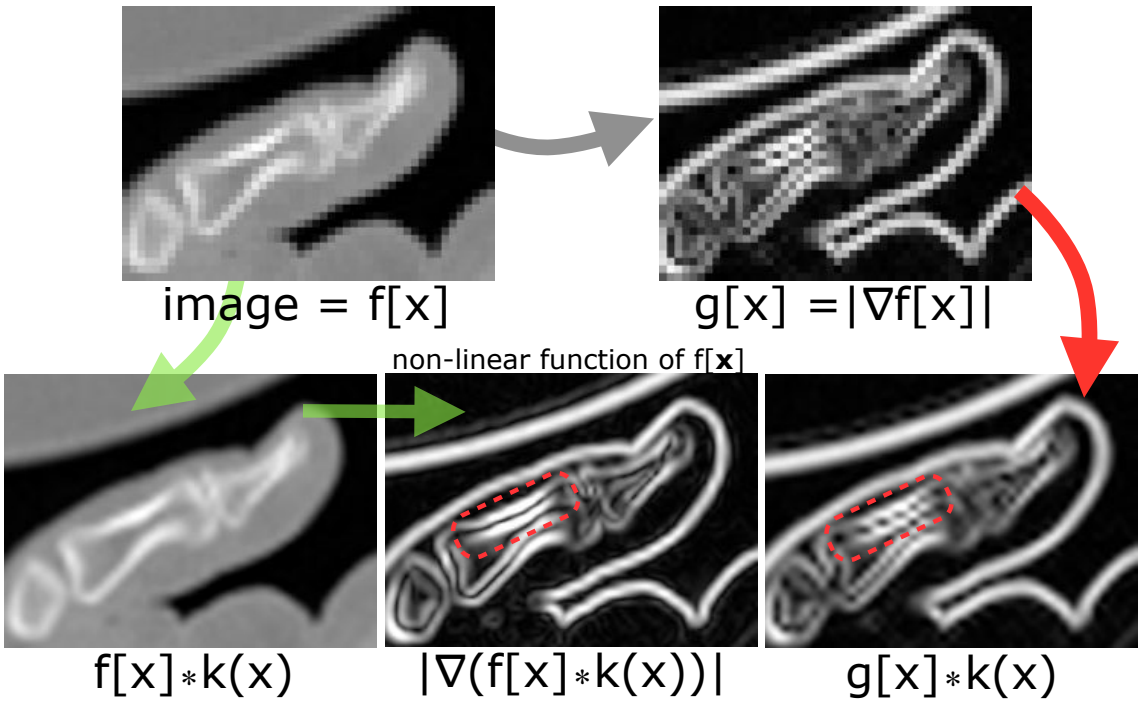
- Goal: geometric models of **meaningful features** for quantitative analysis of large image datasets
  - "Feature" = proxy for structure under scientific study
- Want: **generality** WRT feature co-dimension
- Method: particle systems for feature **sampling**
- But: how do we know if some **mathematical** 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



What I mean by mathematical "feature"

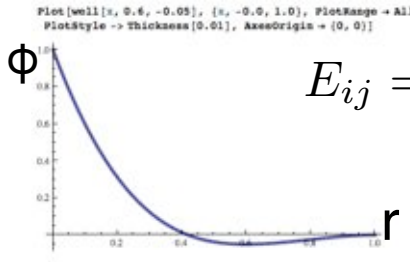
- In a continuous & differentiable field f(**x**) created by convolution:
- Feature = positions **x** satisfying feature equation
  - e.g. Isocontours: $f(\mathbf{x}) = f_0$
  - e.g. Laplacian zero-crossing edges: $\nabla^2 f = 0$
- (No models, priors, blend of data & smoothing terms..)
- ($\mathbf{g} = \nabla f$; $\mathbf{H} = \nabla \otimes \nabla f$; $\mathbf{H} = \sum_i \lambda_i \mathbf{e}_i \otimes \mathbf{e}_i$; $\lambda_1 \geq \lambda_2 \geq \lambda_3$)
- Combinations of maxima and minima WRT $\mathbf{e}_i$
  - Ridge Surface (Eberly '96): $\mathbf{g} \cdot \mathbf{e}_3 = 0$; $\lambda_3 < 0$
  - Ridge Line: $\mathbf{g} \cdot \mathbf{e}_3 = \mathbf{g} \cdot \mathbf{e}_2 = 0$; $\lambda_3 < 0$; $\lambda_2 < 0$
  - Iterative update scheme (Newton optimization) to move closer to feature if near it: we can sample features

Dynamic Particle Systems

- GL Kindlmann, R San Jose Estepar, SM Smith, C-F Westin, **Sampling and Visualizing Creases with Scale-Space Particles**. IEEE Trans. Vis. Comp. Graph, 15(6):1415-1424 (2009)
- Set of points subject to (particle-image) feature constraint and (interparticle) energy minimization

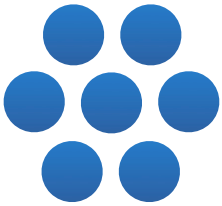
$$\underset{\mathbf{x}_i, N}{\operatorname{argmin}} \mathcal{E} = \underset{\mathbf{x}_i, N}{\operatorname{argmin}} \sum_{i,j=1}^{N} E_{ij}$$

- Hard constraint of particles to feature; no energy



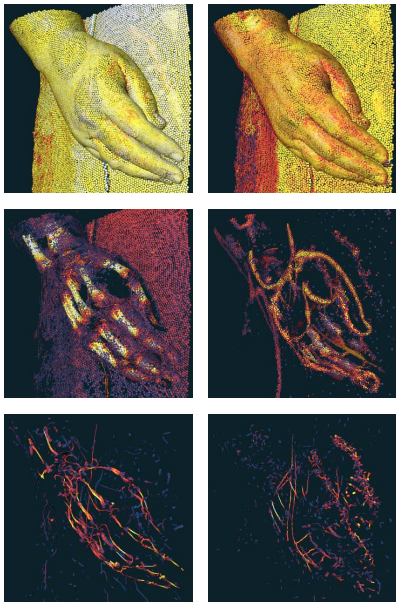
$$E_{ij} = \Phi\left(\frac{|\mathbf{x}_i - \mathbf{x}_j|}{\sigma_r}\right)$$

(demo)



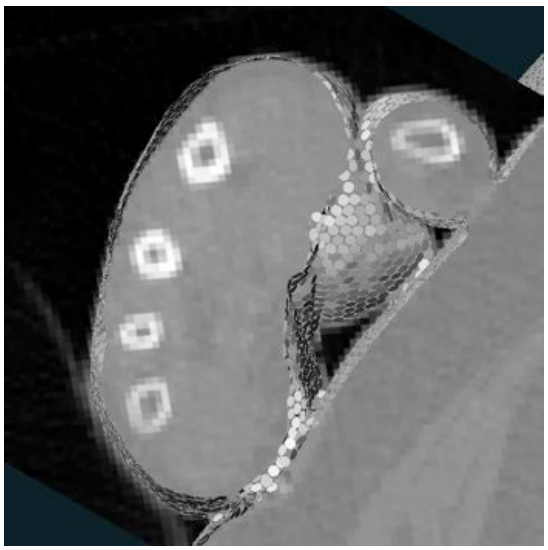
Visual survey of feature types

- 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

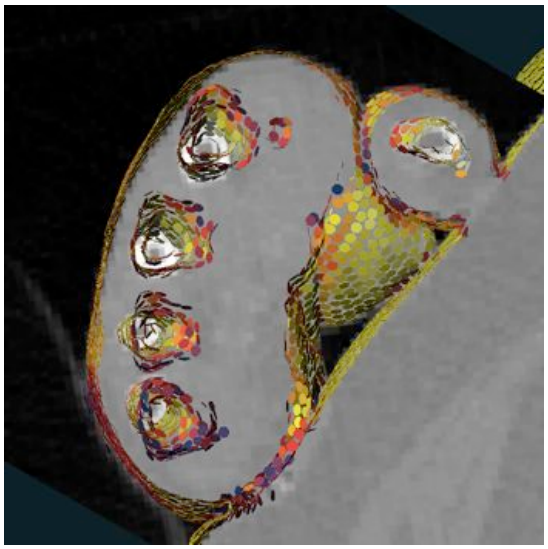
Outside 3D view



Different, Cropped 3D view w/ 2D plane

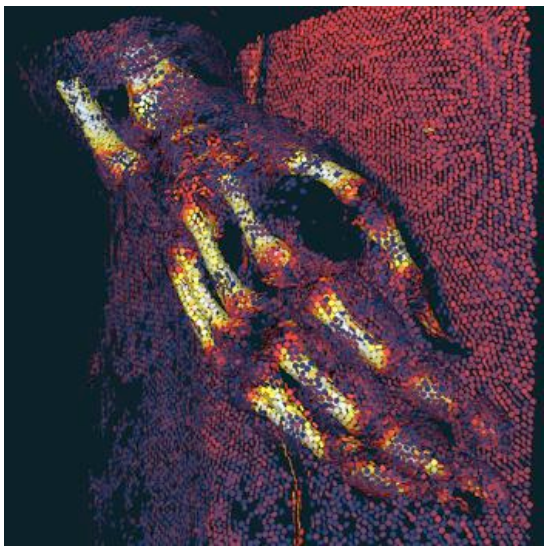
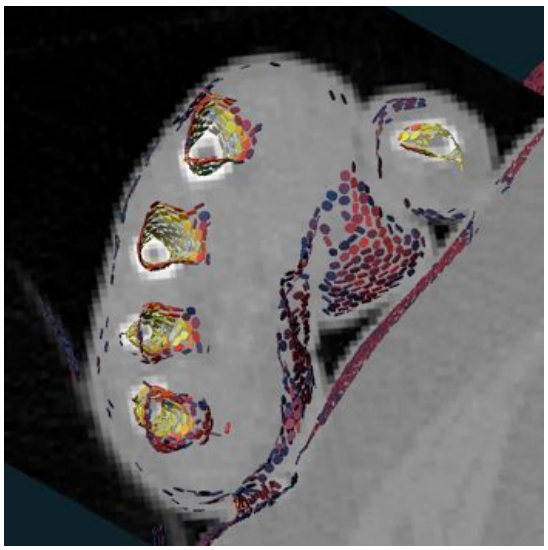
- AKA isocline, isophote, isocontour, level set
- $f(\mathbf{x}) = v_0$

Laplacian 0-crossing

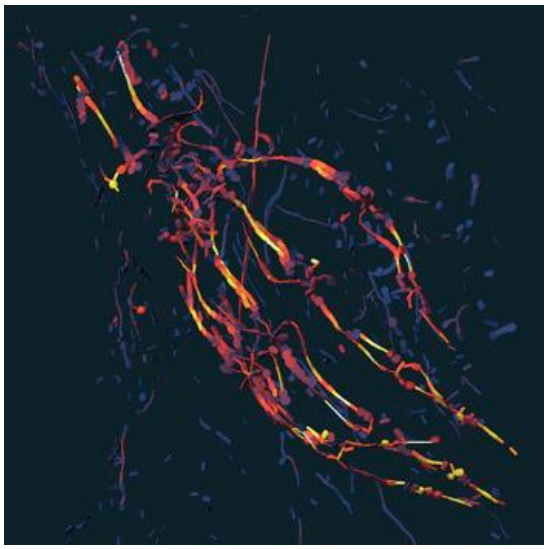
- Classical definition of edge
- $\nabla^2 f(\mathbf{x}) = 0$; strength = $|\nabla f(\mathbf{x})|$

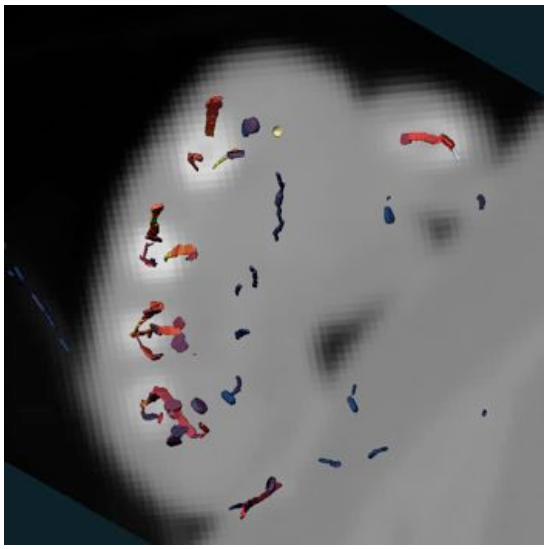
Ridge Surface

- Maximal surface wrt Hessian minor eigenvector $\mathbf{e}_3$
- $\nabla f(\mathbf{x}) \cdot \mathbf{e}_3(\mathbf{x}) = 0$; strength = $-\lambda_3$

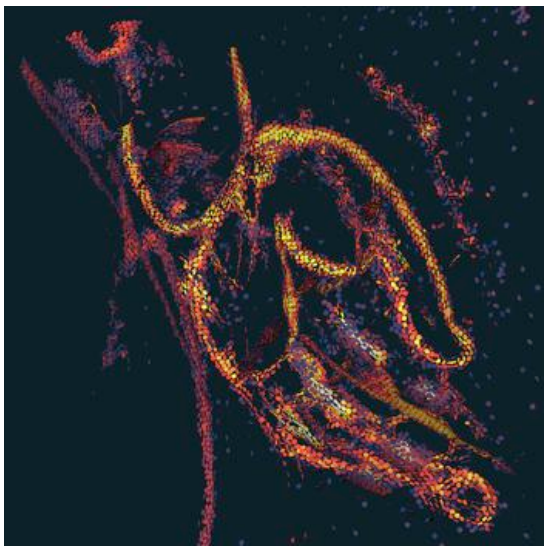
Ridge Line

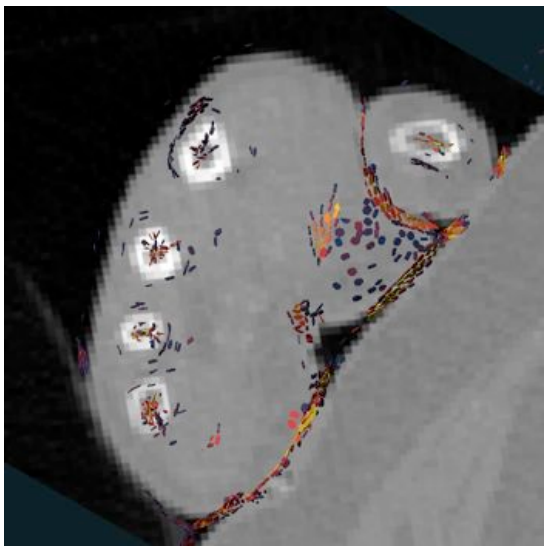




- Maximal curve wrt Hessian minor,medium eigenvectors
- $\nabla f(\mathbf{x}) \cdot \mathbf{e}_3(\mathbf{x}) = 0$, $\nabla f(\mathbf{x}) \cdot \mathbf{e}_2(\mathbf{x}) = 0$; strength = $-\lambda_2$

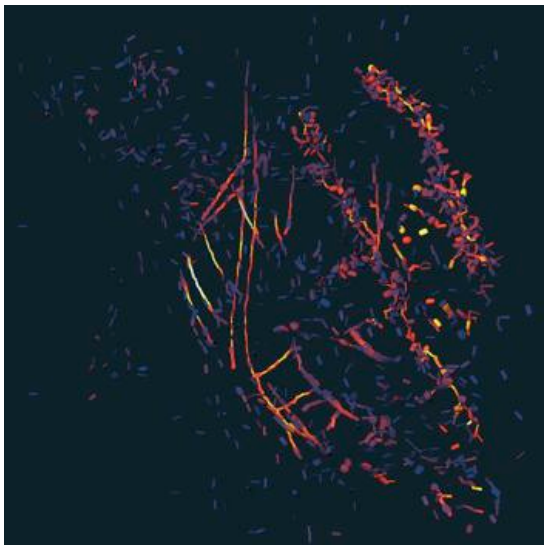
Valley Surface

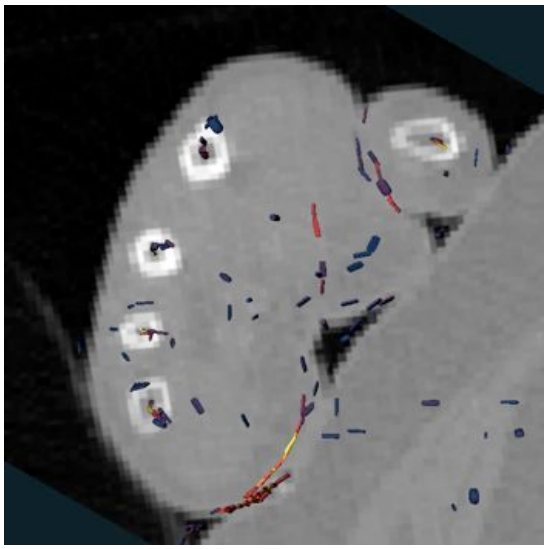




- Minimal surface wrt Hessian major eigenvector $\mathbf{e}_1$
- $\nabla f(\mathbf{x}) \cdot \mathbf{e}_1(\mathbf{x}) = 0$; strength = λ_1

Valley Line

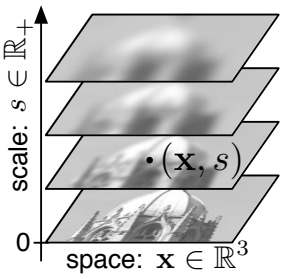




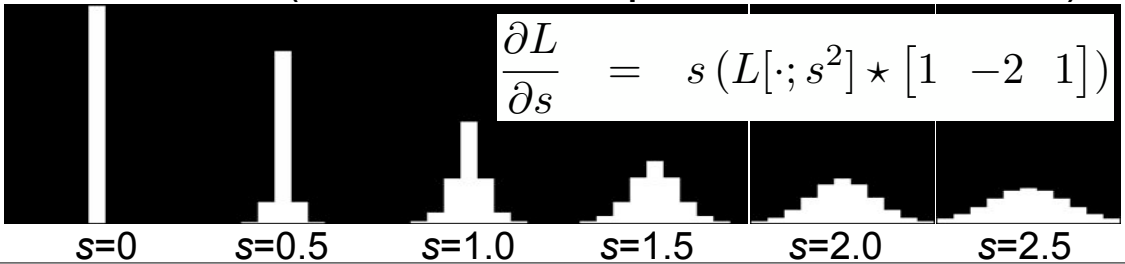
- Minimal curve wrt Hessian major,medium eigenvectors
- $\nabla f(\mathbf{x}) \cdot \mathbf{e}_1(\mathbf{x}) = 0$, $\nabla f(\mathbf{x}) \cdot \mathbf{e}_2(\mathbf{x}) = 0$; strength = λ_2

Scale Space

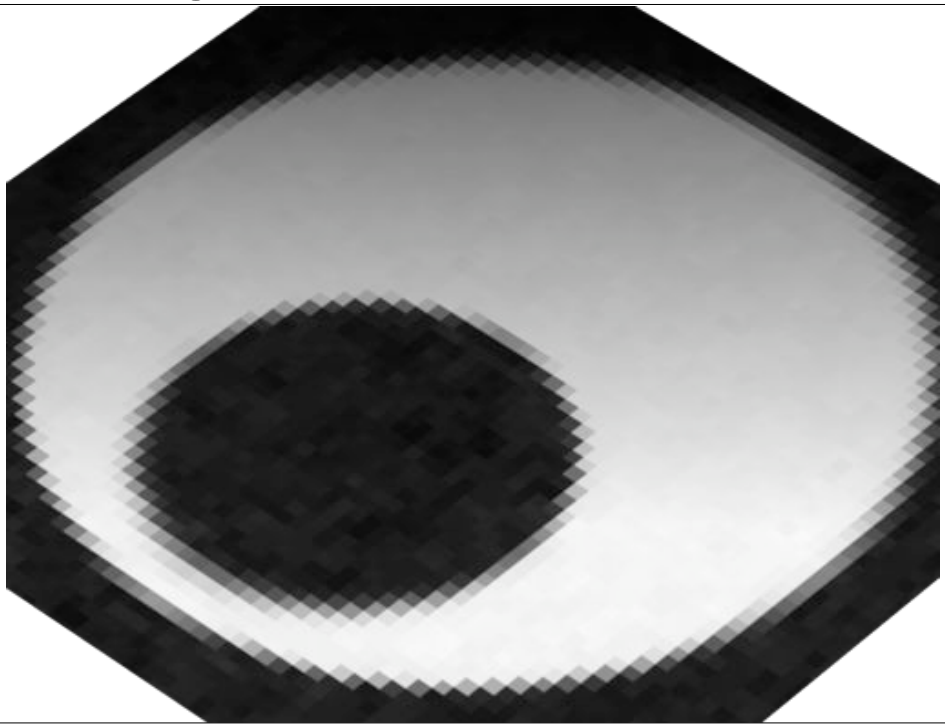
- Image + continuous family of blurrings
  - feature scale not always known a priori
- Practical requirements
  - Probe at arbitrary points in scale space
  - Efficiently handle real-world 3D datasets



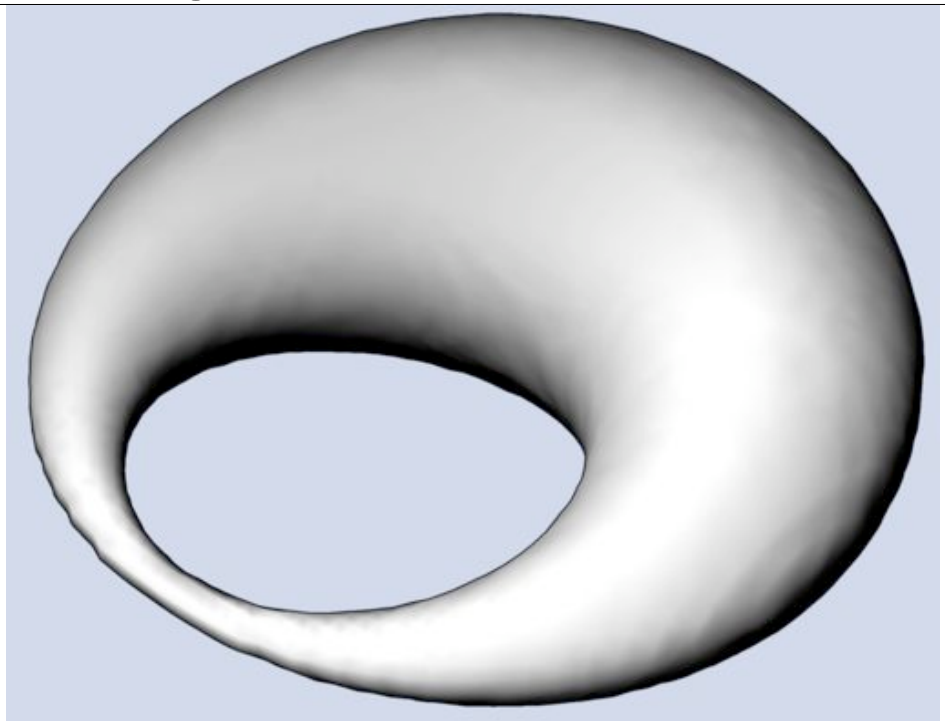
- Scale interpolation based on Lindeberg's "Gaussian" (soln. to heat eq. in discrete domain)



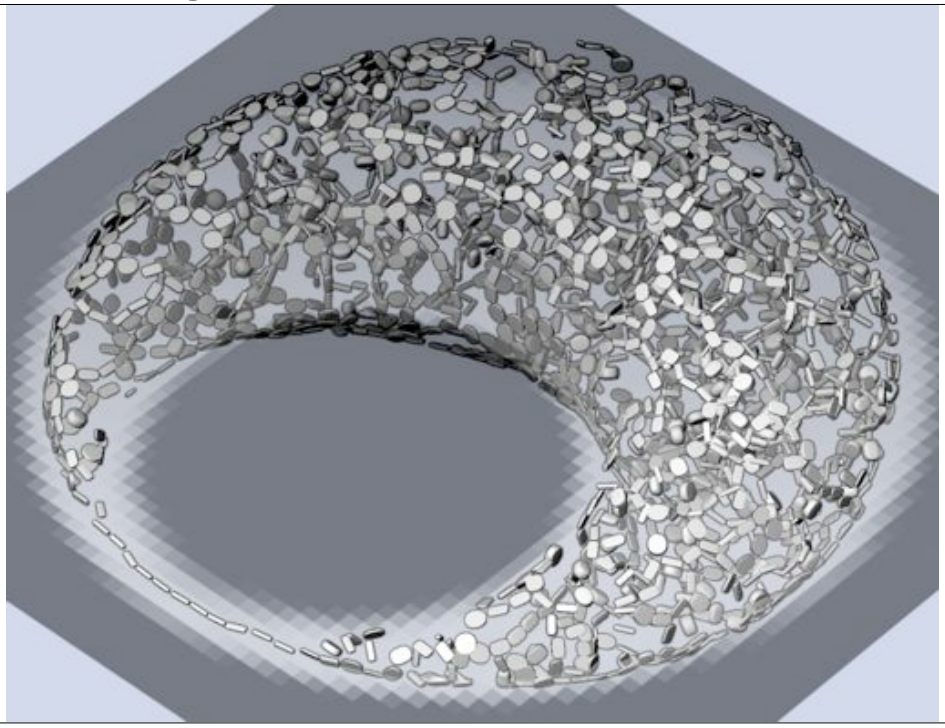
Sampling scale-space feature



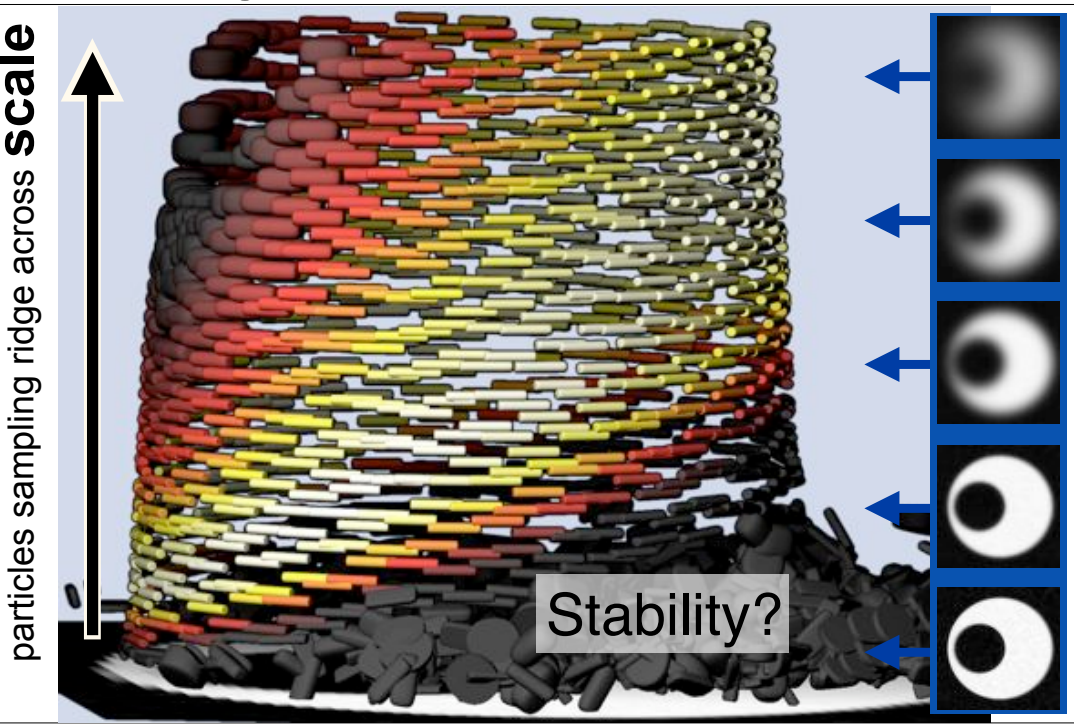
Sampling scale-space feature



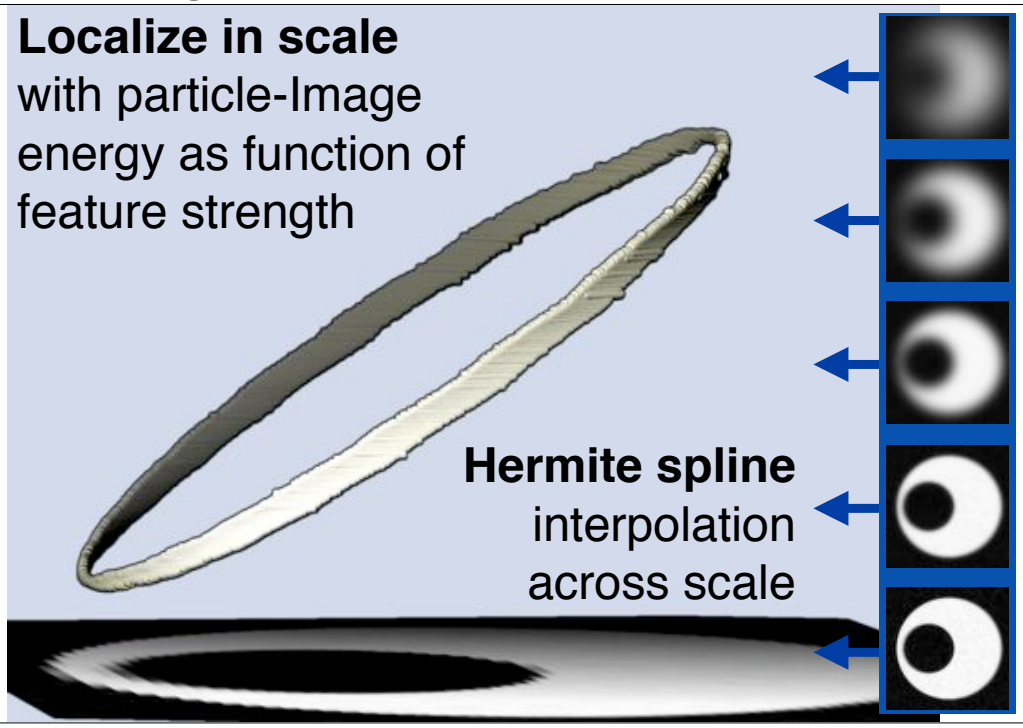
Sampling scale-space feature



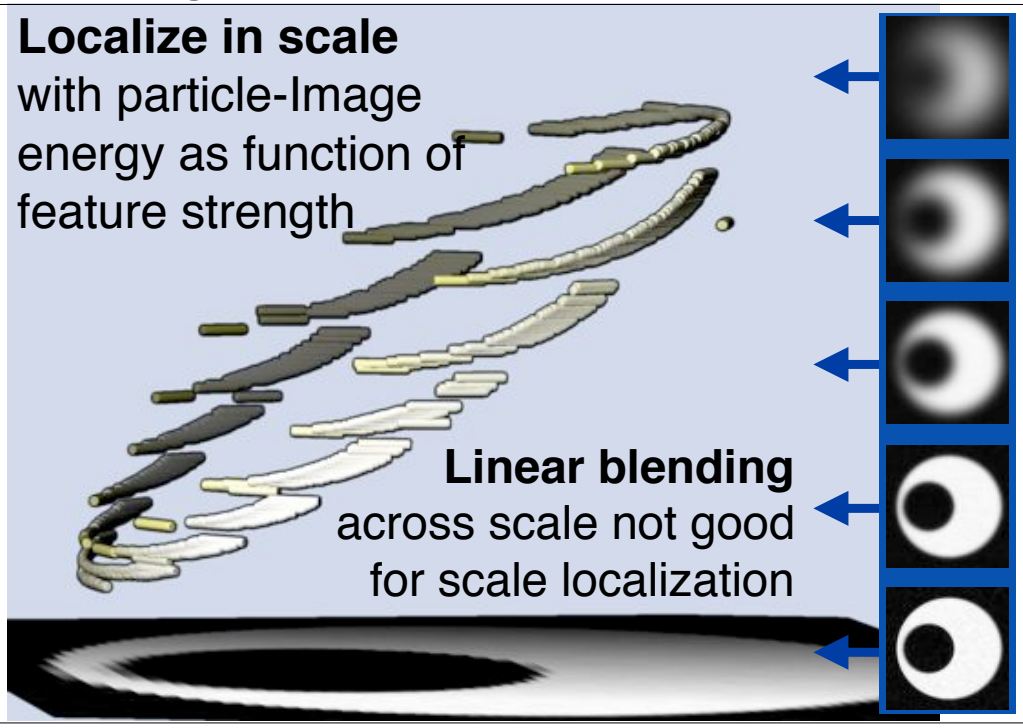
Sampling scale-space feature

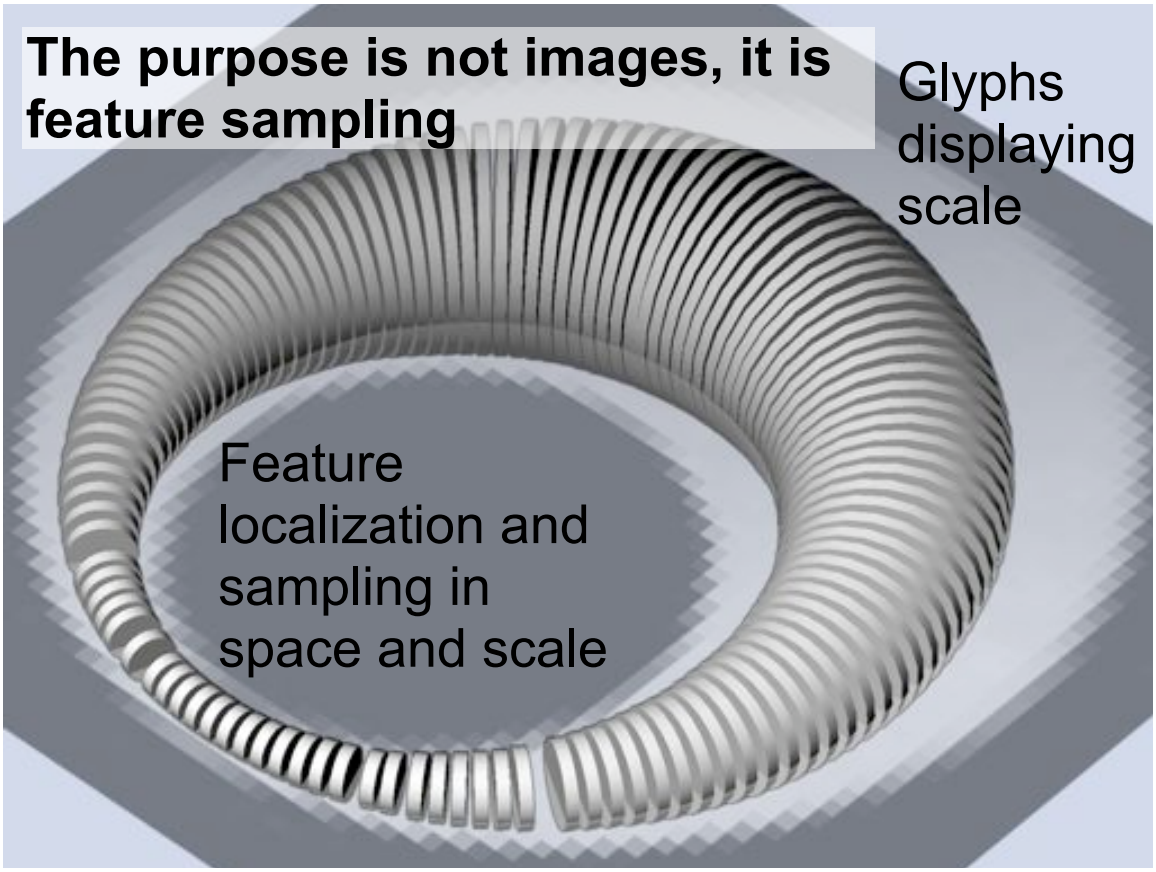


Sampling scale-space feature



Sampling scale-space feature





Applications

Computed Tomographic Measures of Pulmonary Vascular Morphology in Smokers and Their Clinical Implications

Raúl San José Estépar[1,2,3], Gregory L. Kinney[4], Jennifer L. Black-Shinn[4], Russell P. Bowler[5], Gordon L. Kindlmann[6], James C. Ross[3,7], Ron Kiki… Díaz[7], Michael H. Cho[7,9], Craig P. Hersh[7,9], Joyce D… D. Crapo[12], J. Michael Wells[13], Mark T. Dransfield[13], John… ene Study

[1]Department of Radiology, [2]Su… ivision of Pulmonary and Critical Care, and [3]Channing Division of Network … Boston, Massachusetts; [4]Department of Epidemiology, Colorado School … Pulmonary and Critical Care Medicine, and [12]Department of Radiology, Na… nce, University of Chicago, Chicago, Illinois; [5]Division of Pulmonary and Criti… of Medicine, University of Pittsburgh/UPMC, Pittsburgh, Pennsylvania; [12]Dep… nver, Colorado; and [13]Division of Pulmonary and Critical Care, University of …

Rationale: Angiographic invest… cular remodeling in smokers is … blood vessels.
Objectives: Using volumetric … chest we sought to quantitati… its clinical associations.
Methods: Pulmonary vessels … mented, and measured. Tota… aggregate vessel volume for … calculated for all lobes. The lob… ized to the TBV of that lobe and … ($T^{(tissue)}V$) to calculate lobe-spec… Densitometric measures of … Hounsfield unit threshold of …

…IMENTARY

…on the Subject

…ation suggests that pulmonary vascular …s is characterized by distal pruning of …puted tomographic imaging may allow …nts of this process.

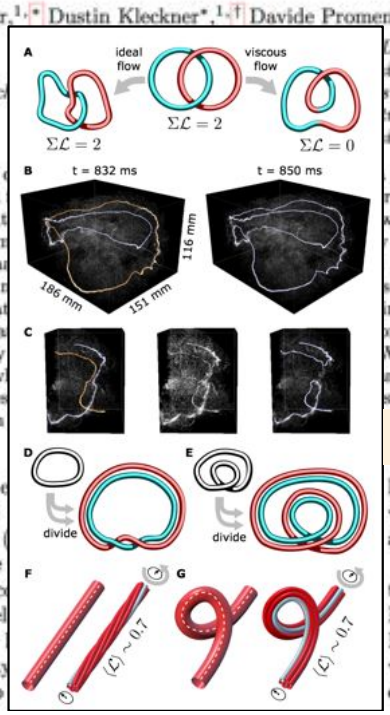
…s to the Field

…monary vascular remodeling in smokers …uning of the intraparenchymal blood

Applications

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

Martin W. Scheeler,[1,*] Dustin Kleckner*,[1,†] Davide Proment,[2] Gordon L. Kindlmann,[3] and William T.M. Irvine[1,‡]



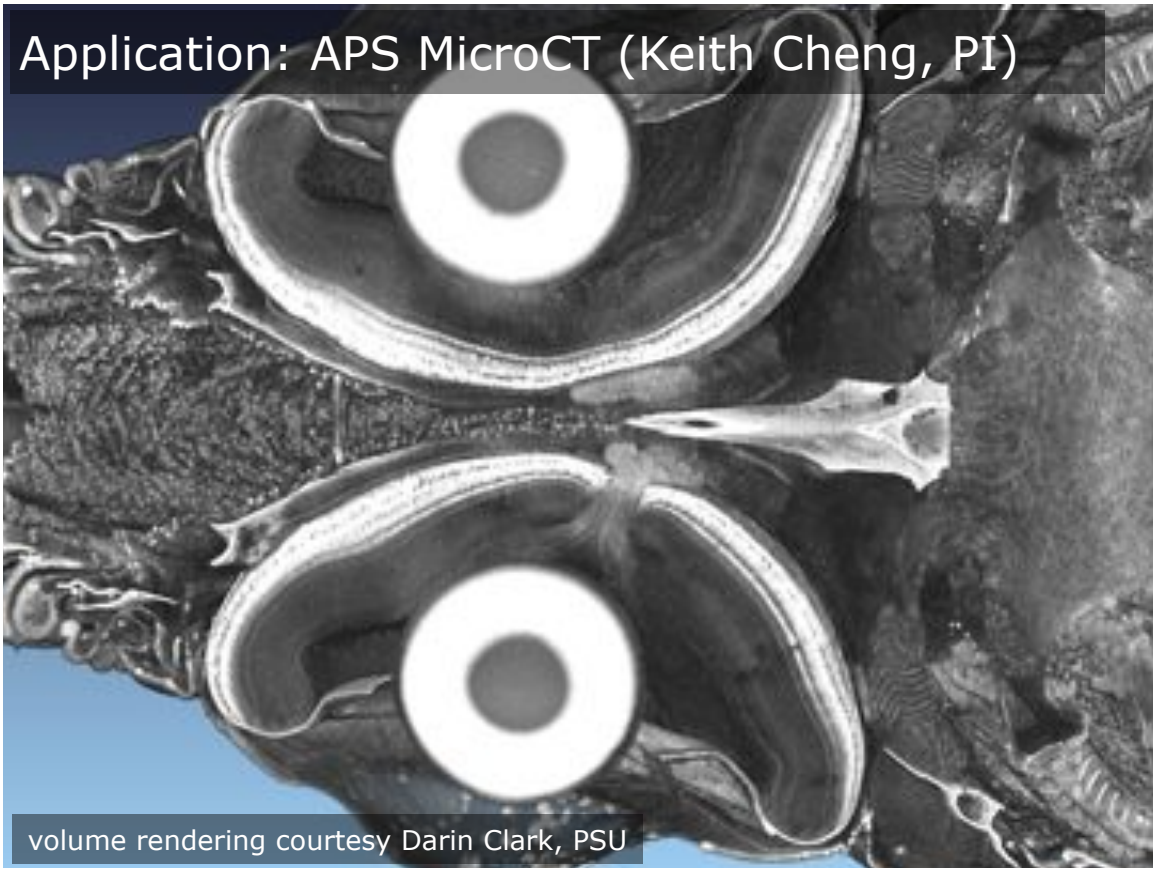
Department of Physics,
icago, Illinois 60637, USA.
t Anglia, NR4 7TJ Norwich, Norfolk, UK.
tment of Computer Science,
icago, Illinois 60637, USA.

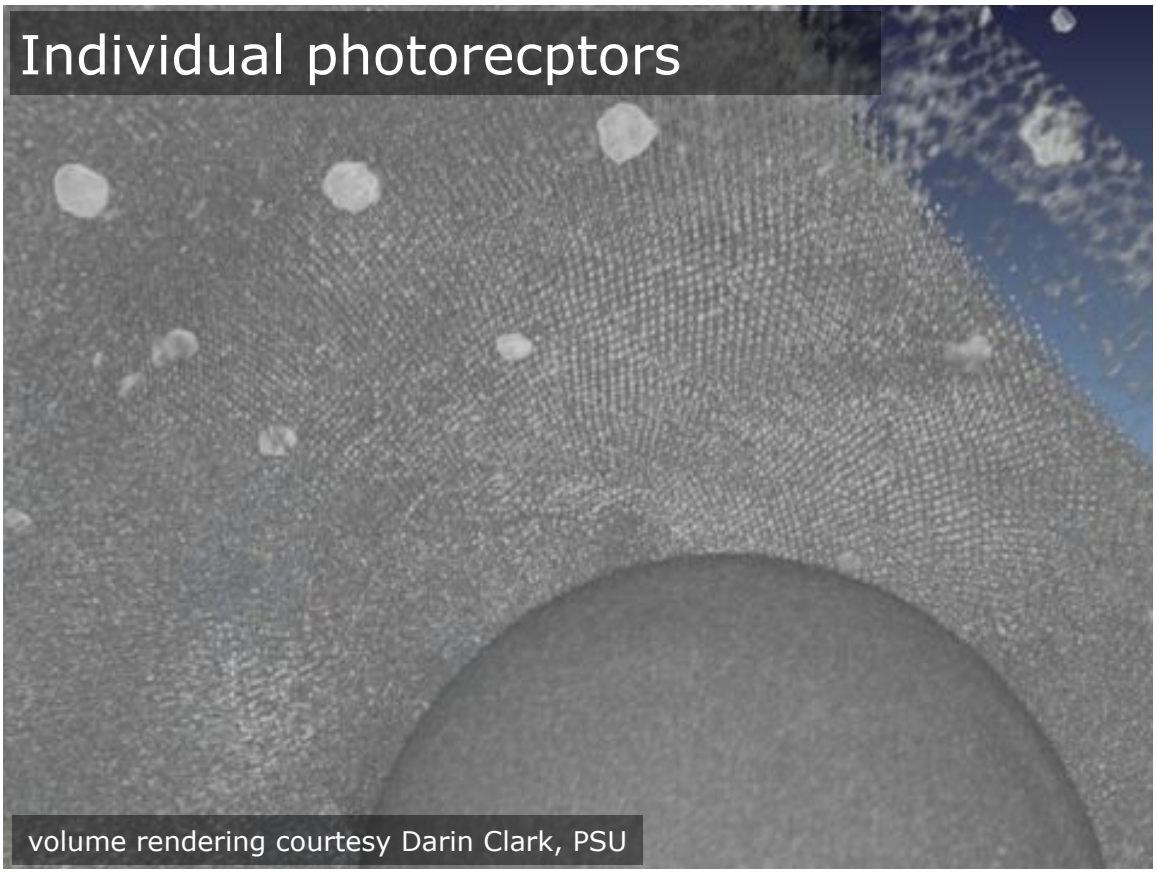
The … fundamental conserved quantity has a rich history in … rvation in the presence of dissipation has proven difficult … we create vortex knots and links in viscous fluids and sim… through topology changing reconnections. We find tha… a reconnection enables the transfer of helicity from lin… remarkably efficient, owing to the anti-parallel orientat… ing vortices. Using a new method for quantifying the spa… nnection process can be viewed as transferring helicity … We also infer the presence of geometric deformations w… ale twist, where it may ultimately be dissipated. Our res… an important role in fluids and related fields, even in …

PNAS 2014 (to appear)

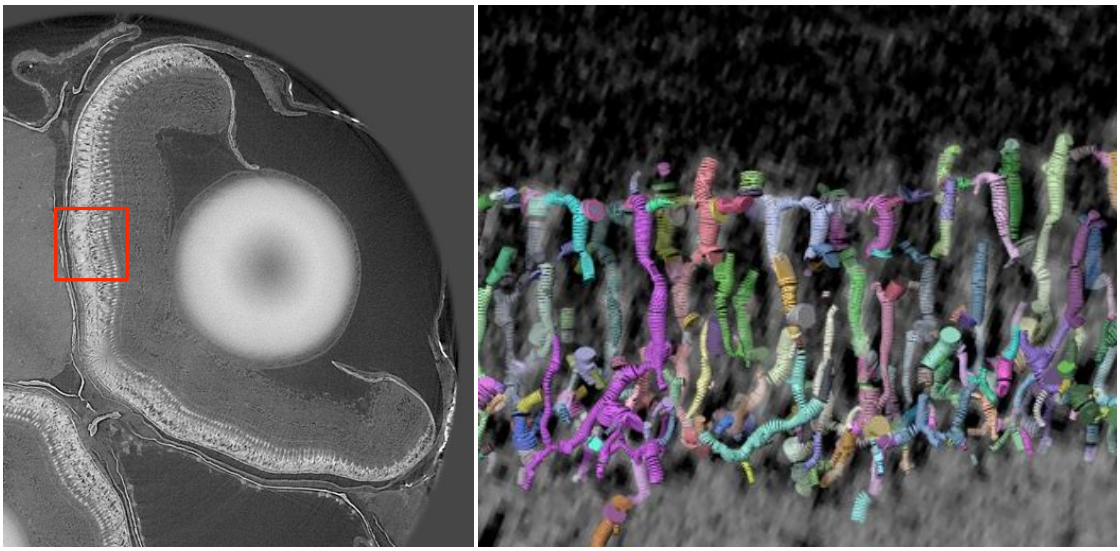
In addition to ene… have been hindered by the lack of techniques to create tum, ideal (Euler) … vortices with topological structure. Thanks to a recent quantity – helicity … advance [19], this is finally possible.

and knotting of the … By performing experiments on linked and knotted vortices in water, as well as numerical simulations of Bose-Einstein condensates (a compressible superfluid [20]) and Biot-Savart vortex evolution, we investigate the conservation of helicity, in so far as it can be inferred from the centerlines of reconnecting vortex tubes. We describe a





Extraction of micro-anatomy



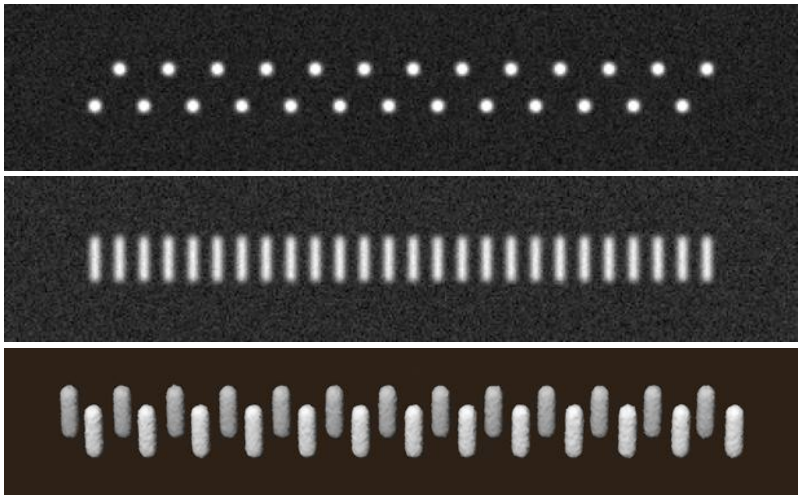
- Extraction of individual photoreceptors from microCT
  - With scale-space particles
  - Lots of structure to be discovered at other scales!

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: its 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

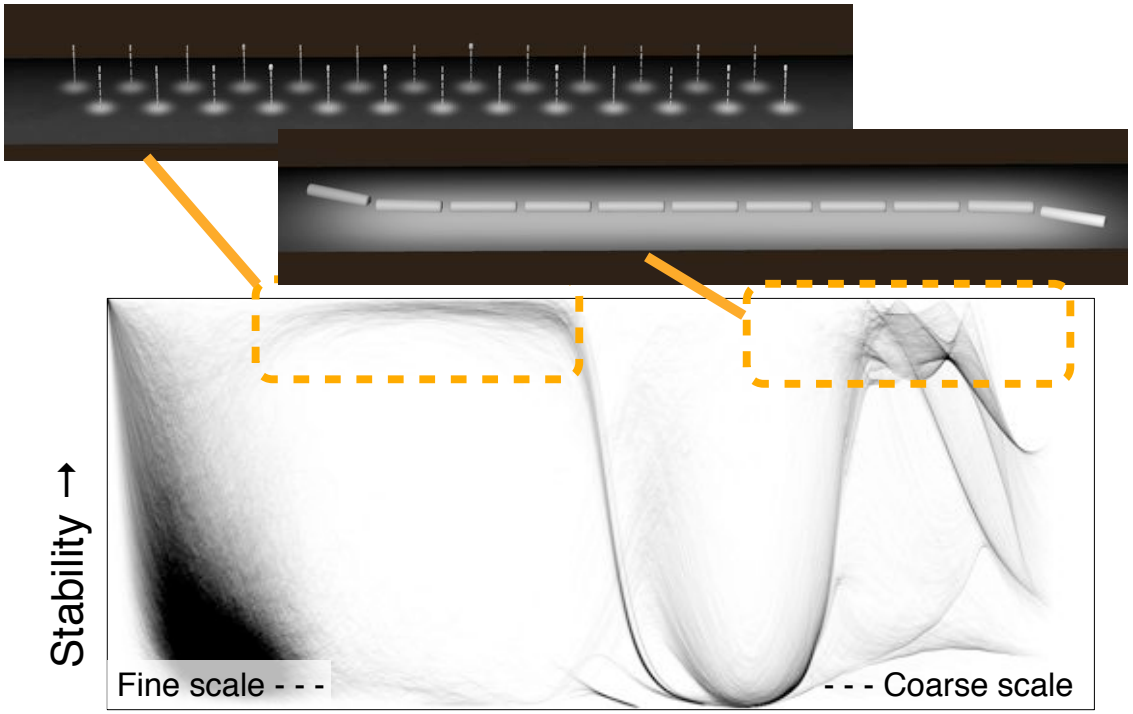


Z slice

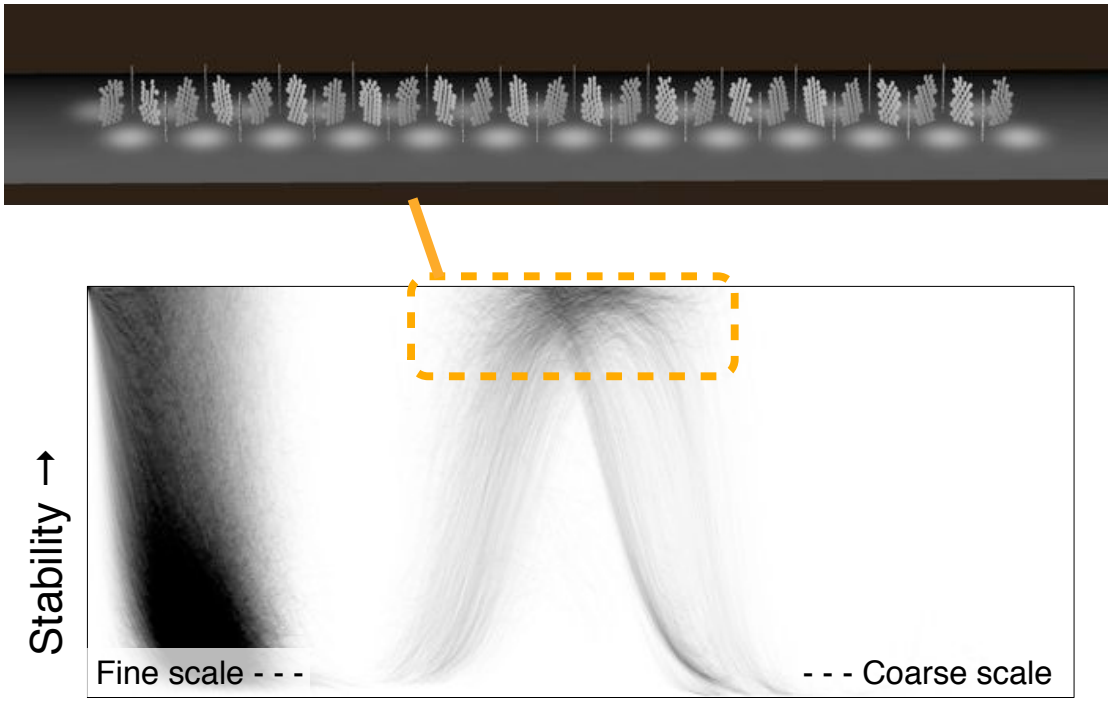
X projection

Isosurface

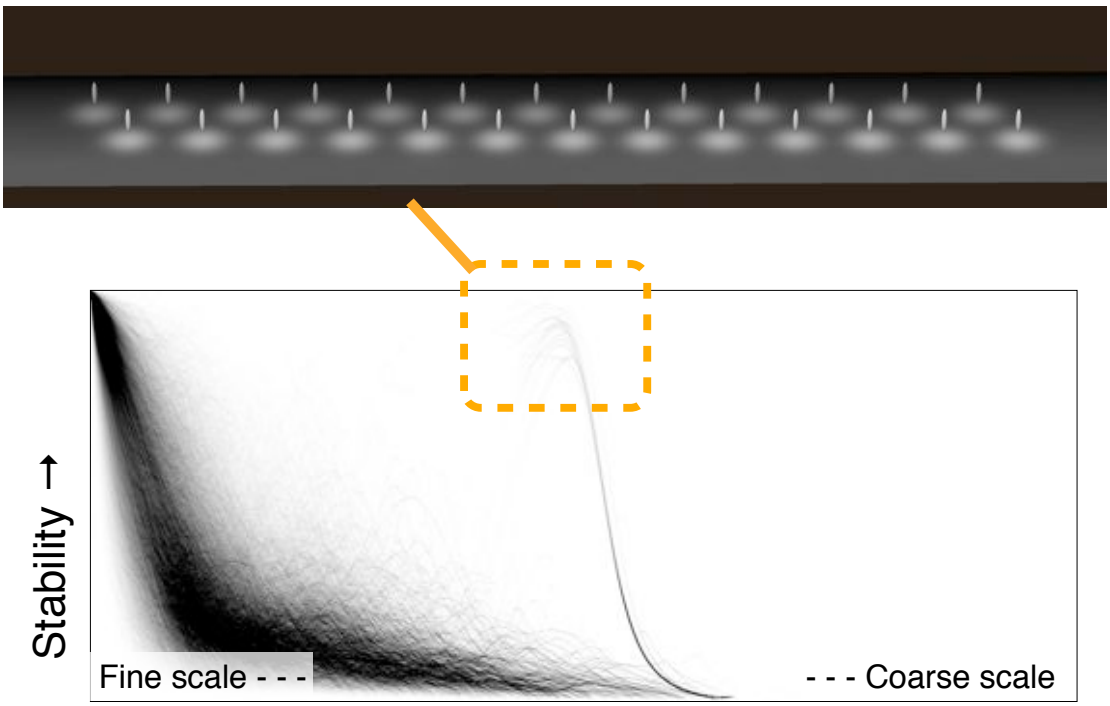
Stability of ridge lines



Stability of valley surfaces



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

(switching gears)

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

Digital Light Sheet Microscopy

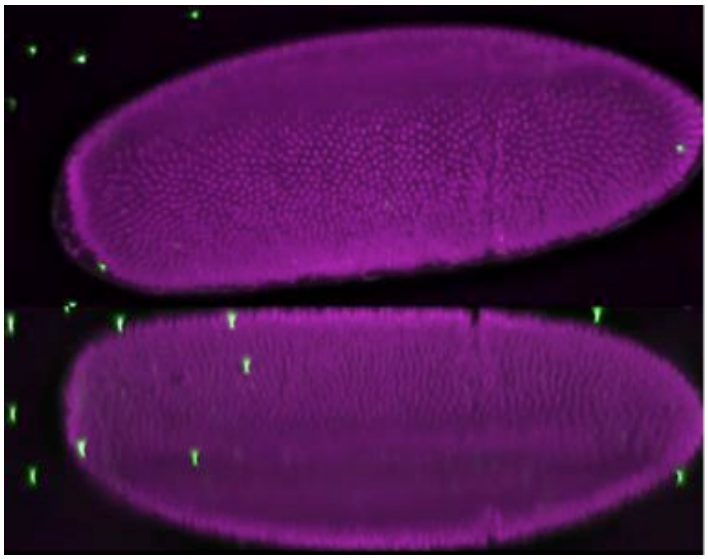
- PI: Kevin White, Institute for Genomics and Systems Biology, U of Chicago

- Drosophila embryo-genesis imaged over 20 hours
- movie shows 2 projections of 3D data

Digital Light Sheet Microscopy

- 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")

- C Chiw, G Kindlmann, J Reppy, L Samuels, and N Seltzer. **Diderot: A Parallel DSL for Image Analysis and Visualization**. In Proceedings of the 33rd ACM SIGPLAN PLDI, pages 111–120, June 2012.

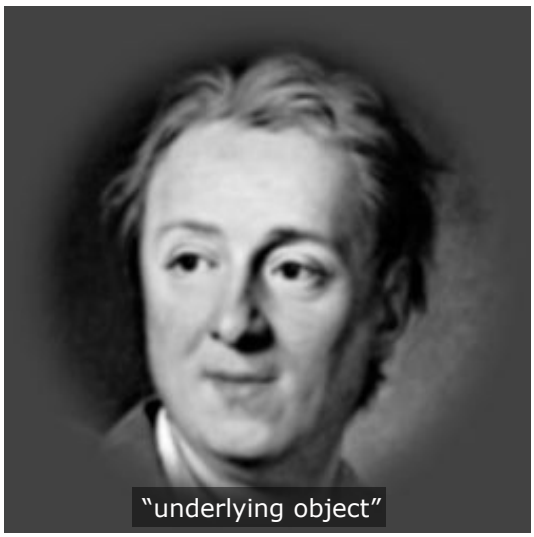
What is Diderot best at?

- 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)

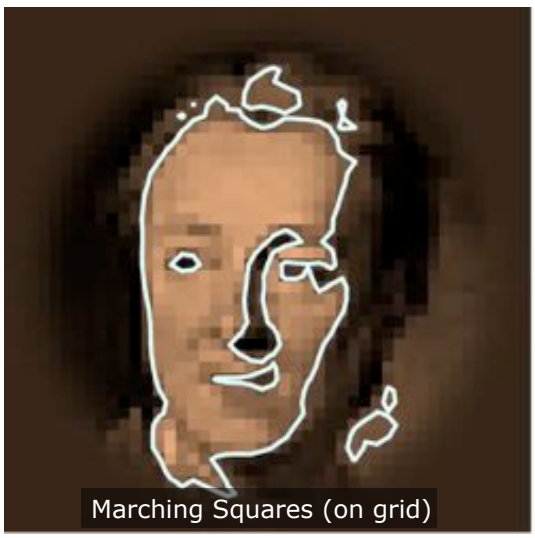
- Measurements of objects produce **images**

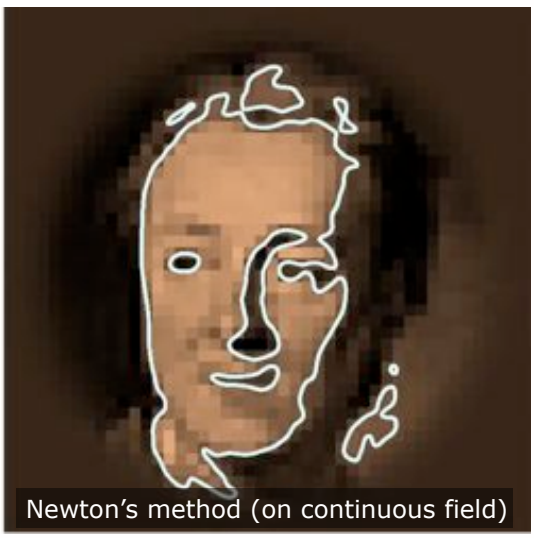


- Goal of scientific vis & analysis is to make statements about the underlying **objects**

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

```
// 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 ];
```

Globals are immutable; used for program inputs

Strands are bulk synchronous

Input parameters for initialization

Strand state, including output

Update method implements algorithm

Initialization of collection of strands with comprehension notation

Execution model

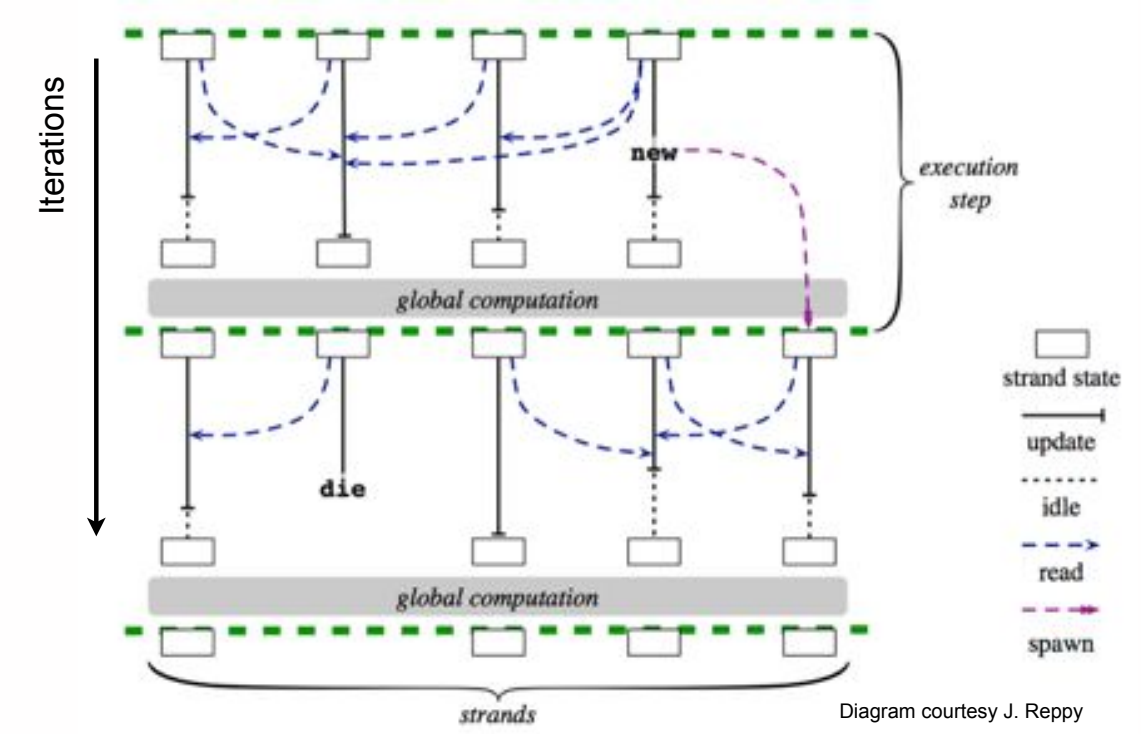
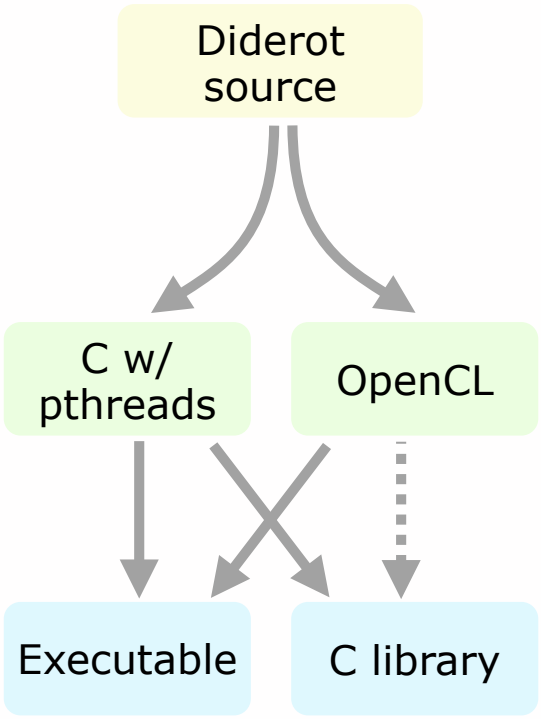


Diagram courtesy J. Reppy

Example: sampling isosurfaces

```
1  input real isoval = 0.4;
2  field#1(2)[] F = ctmr ⊛ image("ddro.nrrd") - isoval;
3  int grid = 150;
4  int stepsMax = 10;
5  real epsilon = 0.000001;
6  strand FindZero(vec2 x0) {
7    output vec2 x = x0;
8    int steps = 0;
9    update {
10     if (!inside(x, F) || steps > stepsMax)
11       die;   // Stop outside domain or after many steps
12     if (|∇F(x)| == 0)
13       die;   // Can't proceed with zero derivative
14     // the Newton-Raphson step
15     vec2 dx = normalize(∇F(x)) * F(x)/|∇F(x)|;
16     x -= dx;
17     if (|dx| < epsilon)
18       stabilize; // Converged when step small enough
19     steps += 1;
20   }
21 }
22 initially { FindZero([lerp(0, 1, -0.5, ui, grid-0.5),
23                       lerp(0, 1, -0.5, vi, grid-0.5)])
24             | vi in 0..(grid-1), ui in 0..(grid-1) };
```

Compilation

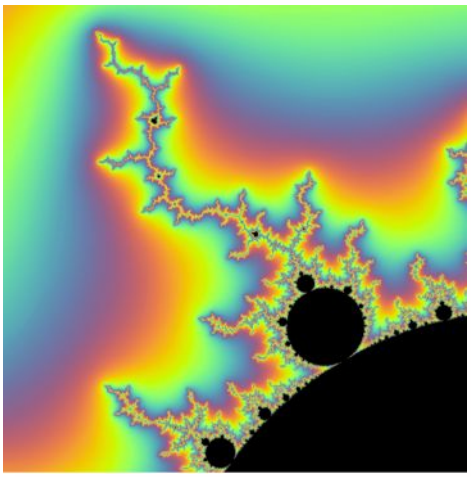


- Compiler written in SML/NJ
- Three stages of intermediate representation

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

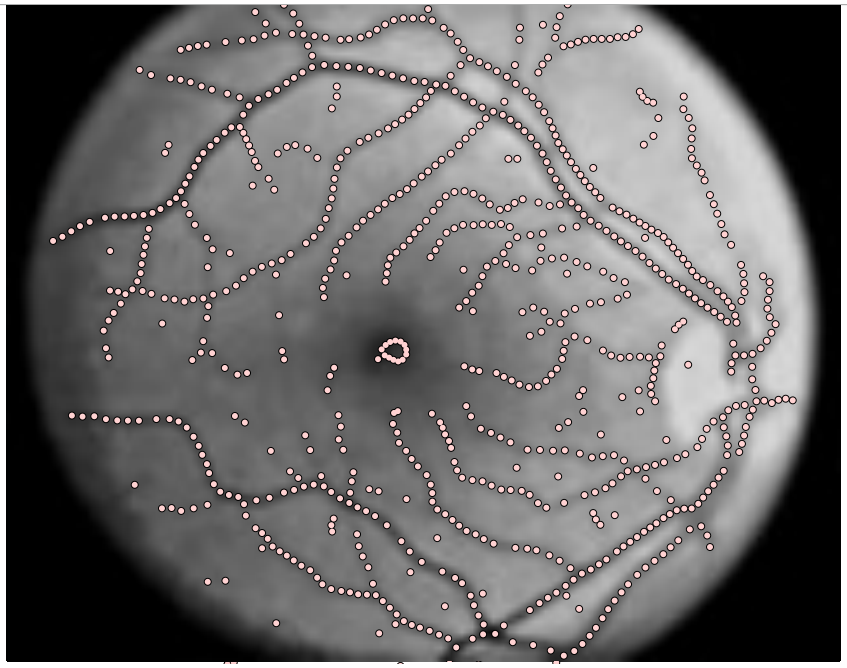
Mandelbrot set

```
// 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#0(1)[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

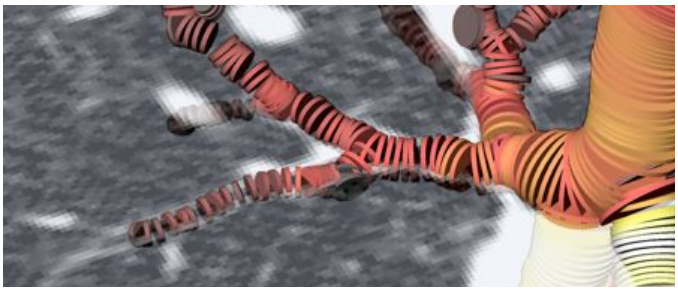


Particles repel each other w/ potential function
(strand communication by Lamont Samuels)

Finding valley lines

```
strand valleyline(vec3 initpos) {
   output vec3 x = initpos;
   update {
      vec3{3} 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;
   }
}
```

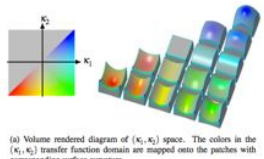
Lung airways (chest CT)



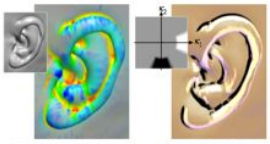
Example visualization method

- Curvature-based transfer functions (Vis'03)

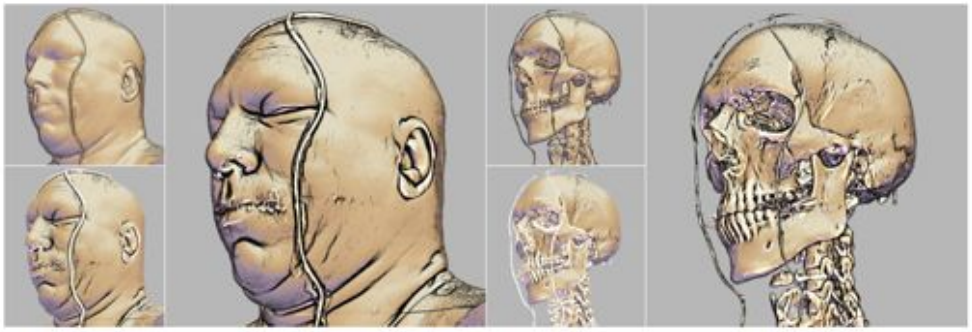
$$
\begin{aligned}
\nabla \mathbf{n}^{\mathrm{T}} &= -\nabla\left(\frac{\mathbf{g}^{\mathrm{T}}}{|\mathbf{g}|}\right) = -\left(\frac{\nabla \mathbf{g}^{\mathrm{T}}}{|\mathbf{g}|} - \frac{\mathbf{g}\,\nabla^{\mathrm{T}}|\mathbf{g}|}{|\mathbf{g}|^2}\right) \\
&= -\frac{1}{|\mathbf{g}|}\left(\mathbf{H} - \frac{\mathbf{g}\,\nabla^{\mathrm{T}}(\mathbf{g}^{\mathrm{T}}\mathbf{g})^{1/2}}{|\mathbf{g}|}\right) = -\frac{1}{|\mathbf{g}|}\left(\mathbf{H} - \frac{\mathbf{g}\,\nabla^{\mathrm{T}}(\mathbf{g}^{\mathrm{T}}\mathbf{g})}{2\,|\mathbf{g}|(\mathbf{g}^{\mathrm{T}}\mathbf{g})^{1/2}}\right) \\
&= -\frac{1}{|\mathbf{g}|}\left(\mathbf{H} - \frac{\mathbf{g}\,(2\mathbf{g}^{\mathrm{T}}\mathbf{H})}{2\,|\mathbf{g}|^2}\right) = -\frac{1}{|\mathbf{g}|}\left(\mathbf{I} - \frac{\mathbf{g}\mathbf{g}^{\mathrm{T}}}{|\mathbf{g}|^2}\right)\mathbf{H} \\
&= -\frac{1}{|\mathbf{g}|}(\mathbf{I} - \mathbf{n}\mathbf{n}^{\mathrm{T}})\mathbf{H}\,.
\end{aligned}
$$



(a) Volume rendered diagram of (κ_1, κ_2) space. The colors in the (κ_1, κ_2) transfer function domain are mapped onto the patches with corresponding surface curvature.



(b) Left: Visualization of ear curvature using transfer function from (a); Right: ridge and valley emphasis implemented with inset transfer function, combined with Gooch shading



The C code to implement that

```
#define DOT_4(a,b) ((a)[0]*(b)[0]+(a)[1]*(b)[1]+(a)[2]*(b)[2]+(a)[3]*(b)[3])
#define VL_4(i, axis) DOT_4(fw0 + (axis)*4, iv##axis + i*4)
#define D1_4(i, axis) DOT_4(fw1 + (axis)*4, iv##axis + i*4)
#define D2_4(i, axis) DOT_4(fw2 + (axis)*4, iv##axis + i*4)

  /* x0 */
  ivY[ 0] = VL_4( 0,X);
  ivY[ 1] = VL_4( 1,X);
  ivY[ 2] = VL_4( 2,X);
  ivY[ 3] = VL_4( 3,X);
  ivY[ 4] = VL_4( 4,X);
  ivY[ 5] = VL_4( 5,X);
  ivY[ 6] = VL_4( 6,X);
  ivY[ 7] = VL_4( 7,X);
  ivY[ 8] = VL_4( 8,X);
  ivY[ 9] = VL_4( 9,X);
  ivY[10] = VL_4(10,X);
  ivY[11] = VL_4(11,X);
  ivY[12] = VL_4(12,X);
  ivY[13] = VL_4(13,X);
  ivY[14] = VL_4(14,X);
  ivY[15] = VL_4(15,X);
  /* x0y0 */
  ivZ[ 0] = VL_4( 0,Y);
  ivZ[ 1] = VL_4( 1,Y);
  ivZ[ 2] = VL_4( 2,Y);
  ivZ[ 3] = VL_4( 3,Y);
  /* x0y0z0 */
  if (doV) {
    *val = VL_4( 0,Z);                          /* f */
  }

  if (!( doD1 || doD2 ))
    return;

  /* x0y0z1 */
  if (doD1) {
    gvec[2] = D1_4( 0,Z);                       /* g_z */
  }
  if (doD2) {
    /* x0y0z2 */
    hess[8] = D2_4( 0,Z);                       /* h_zz */
  }
```

```
ens)) {

ine the sense of the

  /* x0y1 */
  ivZ[ 0] = D1_4( 0,Y);
  ivZ[ 1] = D1_4( 1,Y);
  ivZ[ 2] = D1_4( 2,Y);
  ivZ[ 3] = D1_4( 3,Y);
  /* x0y1z0 */
  if (doD1) {
    gvec[1] = VL_4( 0,Z);                       /* g_y */
  }
  if (doD2) {
    /* x0y1z1 */
    hess[5] = hess[7] = D1_4( 0,Z);             /* h_yz */
    /* x0y2 */
    ivZ[ 0] = D2_4( 0,Y);
    ivZ[ 1] = D2_4( 1,Y);
    ivZ[ 2] = D2_4( 2,Y);
    ivZ[ 3] = D2_4( 3,Y);
    /* x0y2z0 */
    hess[4] = VL_4( 0,Z);                       /* h_yy */
  }
  /* x1 */
  ivY[ 0] = D1_4( 0,X);
  ivY[ 1] = D1_4( 1,X);
  ivY[ 2] = D1_4( 2,X);
  ivY[ 3] = D1_4( 3,X);
  ivY[ 4] = D1_4( 4,X);
  ivY[ 5] = D1_4( 5,X);
  ivY[ 6] = D1_4( 6,X);
  ivY[ 7] = D1_4( 7,X);
  ivY[ 8] = D1_4( 8,X);
  ivY[ 9] = D1_4( 9,X);
  ivY[10] = D1_4(10,X);
  ivY[11] = D1_4(11,X);
  ivY[12] = D1_4(12,X);
  ivY[13] = D1_4(13,X);
  ivY[14] = D1_4(14,X);
  ivY[15] = D1_4(15,X);
  /* x1y0 */
```

OpenCL code (for GPUs)

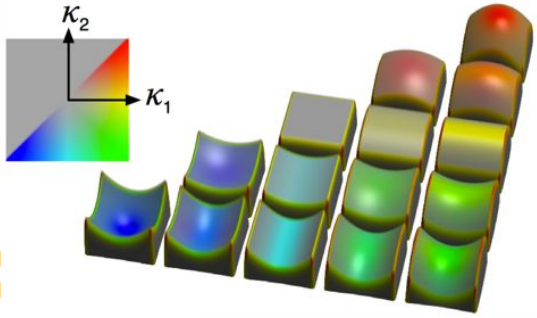
```
float4 computeGradient(image3d_t sampler, float4 gradPos, const float gradOffset)
{
  //central differences gradient
  return (float4)(
            read_imagef(sampler, linearSampler, (float4)(gradPos.x+gradOffset, gradPos.y, gradPos.z, 0.f)).x-
read_imagef(sampler, linearSampler, (float4)(gradPos.x-gradOffset, gradPos.y, gradPos.z, 0.f)).x,
            read_imagef(sampler, linearSampler, (float4)(gradPos.x, gradPos.y+gradOffset, gradPos.z, 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, gradPos.y, gradPos.z+gradOffset, 0.f)).x-
read_imagef(sampler, linearSampler, (float4)(gradPos.x, gradPos.y, gradPos.z-gradOffset, 0.f)).x,
  0.f);
}

float2 computeCurvature(
            image3d_t sampler,
            float4 gradPos,
            const float gradOffset
)
{
  float4 gradient = computeGradient(
                                    sampler,
                                    gradPos,
                                    gradOffset);
  float4 gradient1 = computeGradient(
                                sampler,
                              gradPos+(float4)(gradOffset,0.f,0.f,0.f) ,
                              gradOffset);
  float4 gradient2 = computeGradient(
                                    sampler,
                                    gradPos-(float4)(gradOffset,0.f,0.f,0.f) ,
                                    gradOffset);
  float4 gradient3 = computeGradient(
                                    sampler,
                                    gradPos+(float4)(0.f,gradOffset,0.f,0.f) ,
                                    gradOffset);
  float4 gradient4 = computeGradient(
                                    sampler,
                                    gradPos-(float4)(0.f,gradOffset,0.f,0.f) ,
                                    gradOffset);
  float4 gradient5 = computeGradient(
                                    sampler,
                                    gradPos+(float4)(0.f,0.f,gradOffset,0.f) ,
                                    gradOffset);
  float4 gradient6 = computeGradient(
                                    sampler,
                                    gradPos-(float4)(0.f,0.f,gradOffset,0.f) ,
                                    gradOffset);

  gradient1 = fast_normalize(gradient1);
  gradient2 = fast_normalize(gradient2);
  gradient3 = fast_normalize(gradient3);
  gradient4 = fast_normalize(gradient4);
  gradient5 = fast_normalize(gradient5);
  gradient6 = fast_normalize(gradient6);
```

Courtesy Klaus Engel, Siemens

```
  float l = fast_length(gradient);
  if (l == 0.f)
     return (float2)(0.f,0.f);
  float4 n = -gradient/l;
  float P[3][3];
  P[0][0] = 1.f-n.x*n.x;
  P[0][1] = n.x*n.y;
  P[0][2] = n.x*n.z;
  P[1][0] = n.y*n.x;
  P[1][1] = 1.f-n.y*n.y;
  P[1][2] = n.y*n.z;
  P[2][0] = n.z*n.x;
  P[2][1] = n.z*n.y;
  P[2][2] = 1.f-n.z*n.z;

  float hessian[3][3];
  hessian[0][0] = gradient1.x - gradient2.x;
  hessian[0][1] = gradient1.y - gradient2.y;
  hessian[0][2] = gradient1.z - gradient2.z;
  hessian[1][0] = gradient3.x - gradient4.x;
  hessian[1][1] = gradient3.y - gradient4.y;
  hessian[1][2] = gradient3.z - gradient4.z;
  hessian[2][0] = gradient5.x - gradient6.x;
  hessian[2][1] = gradient5.y - gradient6.y;
  hessian[2][2] = gradient5.z - gradient6.z;

  float T[3][3];
  float G[3][3];// = -P*hessian*P/l;

  T[0][0] = -P[0][0]*hessian[0][0] - P[1][0]*hessian[0][1] - P[2][0]*hessian[0][2];
  T[1][0] = -P[0][0]*hessian[1][0] - P[1][0]*hessian[1][1] - P[2][0]*hessian[1][2];
  T[2][0] = -P[0][0]*hessian[2][0] - P[1][0]*hessian[2][1] - P[2][0]*hessian[2][2];
  T[0][1] = -P[0][1]*hessian[0][0] - P[1][1]*hessian[0][1] - P[2][1]*hessian[0][2];
  T[1][1] = -P[0][1]*hessian[1][0] - P[1][1]*hessian[1][1] - P[2][1]*hessian[1][2];
  T[2][1] = -P[0][1]*hessian[2][0] - P[1][1]*hessian[2][1] - P[2][1]*hessian[2][2];
  T[0][2] = -P[0][2]*hessian[0][0] - P[1][2]*hessian[0][1] - P[2][2]*hessian[0][2];
  T[1][2] = -P[0][2]*hessian[1][0] - P[1][2]*hessian[1][1] - P[2][2]*hessian[1][2];
  T[2][2] = -P[0][2]*hessian[2][0] - P[1][2]*hessian[2][1] - P[2][2]*hessian[2][2];

  G[0][0] = (T[0][0]*P[0][0] + T[1][0]*P[0][1] + T[2][0]*P[0][2])/l;
  G[1][0] = (T[0][0]*P[1][0] + T[1][0]*P[1][1] + T[2][0]*P[1][2])/l;
  G[2][0] = (T[0][0]*P[2][0] + T[1][0]*P[2][1] + T[2][0]*P[2][2])/l;
  G[0][1] = (T[0][1]*P[0][0] + T[1][1]*P[0][1] + T[2][1]*P[0][2])/l;
  G[1][1] = (T[0][1]*P[1][0] + T[1][1]*P[1][1] + T[2][1]*P[1][2])/l;
  G[2][1] = (T[0][1]*P[2][0] + T[1][1]*P[2][1] + T[2][1]*P[2][2])/l;
  G[0][2] = (T[0][2]*P[0][0] + T[1][2]*P[0][1] + T[2][2]*P[0][2])/l;
  G[1][2] = (T[0][2]*P[1][0] + T[1][2]*P[1][1] + T[2][2]*P[1][2])/l;
  G[2][2] = (T[0][2]*P[2][0] + T[1][2]*P[2][1] + T[2][2]*P[2][2])/l;

  float t = G[0][0]+G[1][1]+G[2][2];
  float f = sqrt(G[0][0]*G[0][0]+G[0][1]*G[0][1]+G[0][2]*G[0][2]+
                                        G[1][0]*G[1][0]+G[1][1]*G[1][1]+G[1][2]*G[1][2]+
                                        G[2][0]*G[2][0]+G[2][1]*G[2][1]+G[2][2]*G[2][2]);

  float k1 = (t + sqrt(2.f*f-t*t))/2.f;
  float k2 = (t - sqrt(2.f*f-t*t))/2.f;

  return (float2)(k1,k2);
}
```

Example: curvature measurement

```
// volume dataset
field#2(3)[] F = bspln3 ⊛ load("quads.img");

// RGB colormap of (kappa1,kapp2)
field#0(2)[3] RGB = tent ⊛ load("rgb.img");
```




```
vec3 grad = -∇F(pos);
vec3 norm = normalize(grad);
// begin curvature computation
tensor[3,3] H = ∇⊗∇F(pos);
tensor[3,3] P = identity[3] - norm⊗norm;
tensor[3,3] G = -(P•H•P)/|grad|;
real disc = max(0.0, sqrt(2.0*|G|^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
  - **lifting**: field#1(3)[3] V = ctmr ⊛ image(vec.nrrd) → field#1(3)[] M = |V| (Charisee Chiw)
  - **composition**: V = W ◦ F, and then differentiation by chain rule
  - Fields from point clouds, Finite Element Meshes
- Many possibilities for GUI / IDE:
  - Sliders for all **`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