Rendering and Extracting Extremal Features in 3D Fields

Gordon Kindlmann*, Charisee Chiw*, Tri Huynh*, Attila Gyulassy**,
John Reppy*, Peer-Timo Bremer**

*Department of Computer Science, University of Chicago

** SCI Institute, University of Utah

Paper is about Mathematics for Visualization

Taylor series:

$$f(\mathbf{x}+\boldsymbol{\epsilon}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \boldsymbol{\epsilon}$$

$$v_0 = f(\mathbf{x}+\boldsymbol{\epsilon}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \boldsymbol{\epsilon}$$

$$\Rightarrow v_0 - f(\mathbf{x}) \approx \nabla f(\mathbf{x}) \cdot \boldsymbol{\epsilon}$$

$$\Rightarrow v_0 - f(\mathbf{x}) = \nabla f(\mathbf{x}) \cdot d\frac{\nabla f(\mathbf{x})}{|\nabla f(\mathbf{x})|} = d|\nabla f(\mathbf{x})|$$

$$\Rightarrow d = \frac{v_0 - f(\mathbf{x})}{|\nabla f(\mathbf{x})|} \Rightarrow \boldsymbol{\epsilon} = \frac{(v_0 - f(\mathbf{x}))\nabla f(\mathbf{x})}{\nabla f(\mathbf{x}) \cdot \nabla f(\mathbf{x})}$$

Newton step for isosurface

Isosurface:

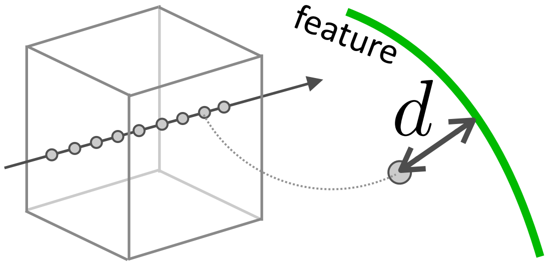
$$f(\mathbf{x}) = v_0$$

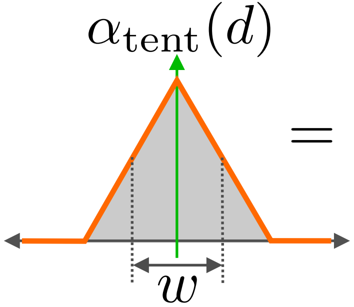
Step from $\mathbf{x}$ to isosurface:

$$\boldsymbol{\epsilon} = d\frac{\nabla f(\mathbf{x})}{|\nabla f(\mathbf{x})|}$$

Newton steps for Visualization?

Direct volume rendering:





$$\alpha_{\text{tent}}(d) = \max\left(0, 1 - \frac{|d|}{w}\right)$$

$$\Rightarrow \alpha(\mathbf{x}) = \alpha_{\text{tent}}(|\boldsymbol{\epsilon}(\mathbf{x})|) = \max\left(0, 1 - \frac{1}{w}\frac{|v_0 - f(\mathbf{x})|}{|\nabla f(\mathbf{x})|}\right)$$

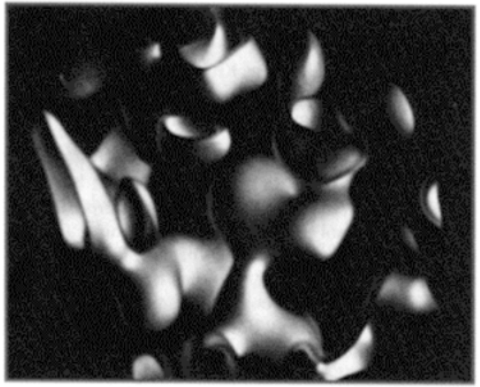


Figure 6. Volume rendering of isovalue contour surface from dataset shown in Figure 5.

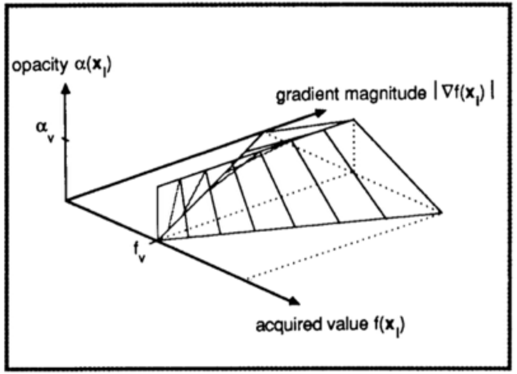


Figure 3. Calculation of opacities for isovalue contour surfaces.

$$\alpha(\mathbf{x}_i) = \alpha_v \begin{cases} 1 & \text{if } |\nabla f(\mathbf{x}_i)| = 0 \text{ and } f(\mathbf{x}_i) = f_v \\ 1 - \dfrac{1}{r}\left|\dfrac{f_v - f(\mathbf{x}_i)}{|\nabla f(\mathbf{x}_i)|}\right| & \text{if } |\nabla f(\mathbf{x}_i)| > 0 \text{ and } f(\mathbf{x}_i) - r\,|\nabla f(\mathbf{x}_i)| \le f_v \le f(\mathbf{x}_i) + r\,|\nabla f(\mathbf{x}_i)| \\ 0 & \text{otherwise} \end{cases}$$

[Levoy-CGnA-1988]

Yes: Newton steps for Visualization!

Vis method

(implementation of main algorithm)

Direct Volume Rendering

Particle-based Feature Extraction

Newton step!

(basic idea of our paper)

Vis target

(mathematical feature of interest)

Isosurfaces

Extremal features
(critical points, ridges, valleys, surface creases)
[Guy-PAMI-1997], [Tang-VIS-1998], [Amenta-SIGGRAPH-2004]

for scalar, and vector, and tensor data

Outline

Basic idea

Synthetic data: 2 vis methods, various features

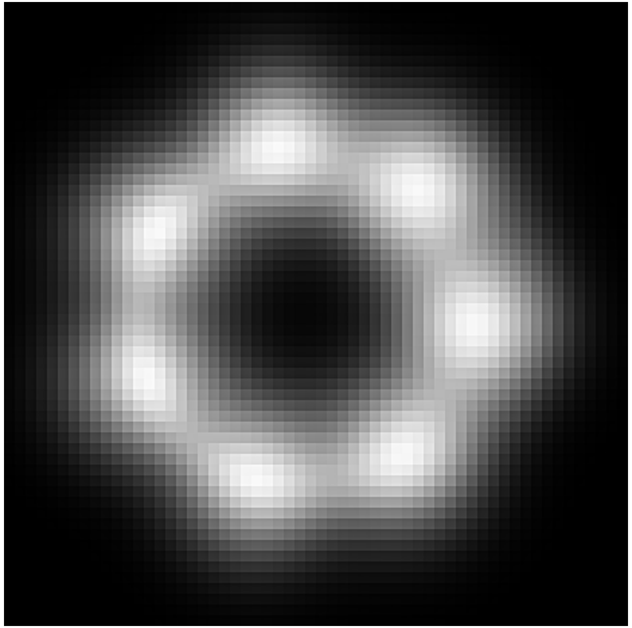
Technical aspects

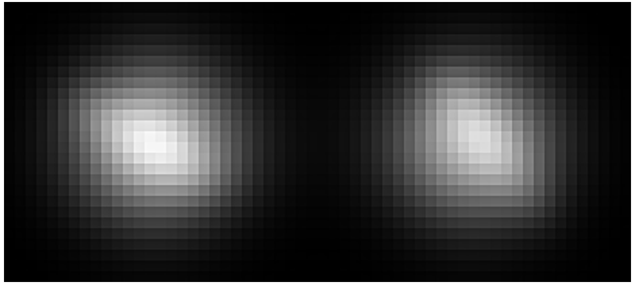
- point meshing, feature strength

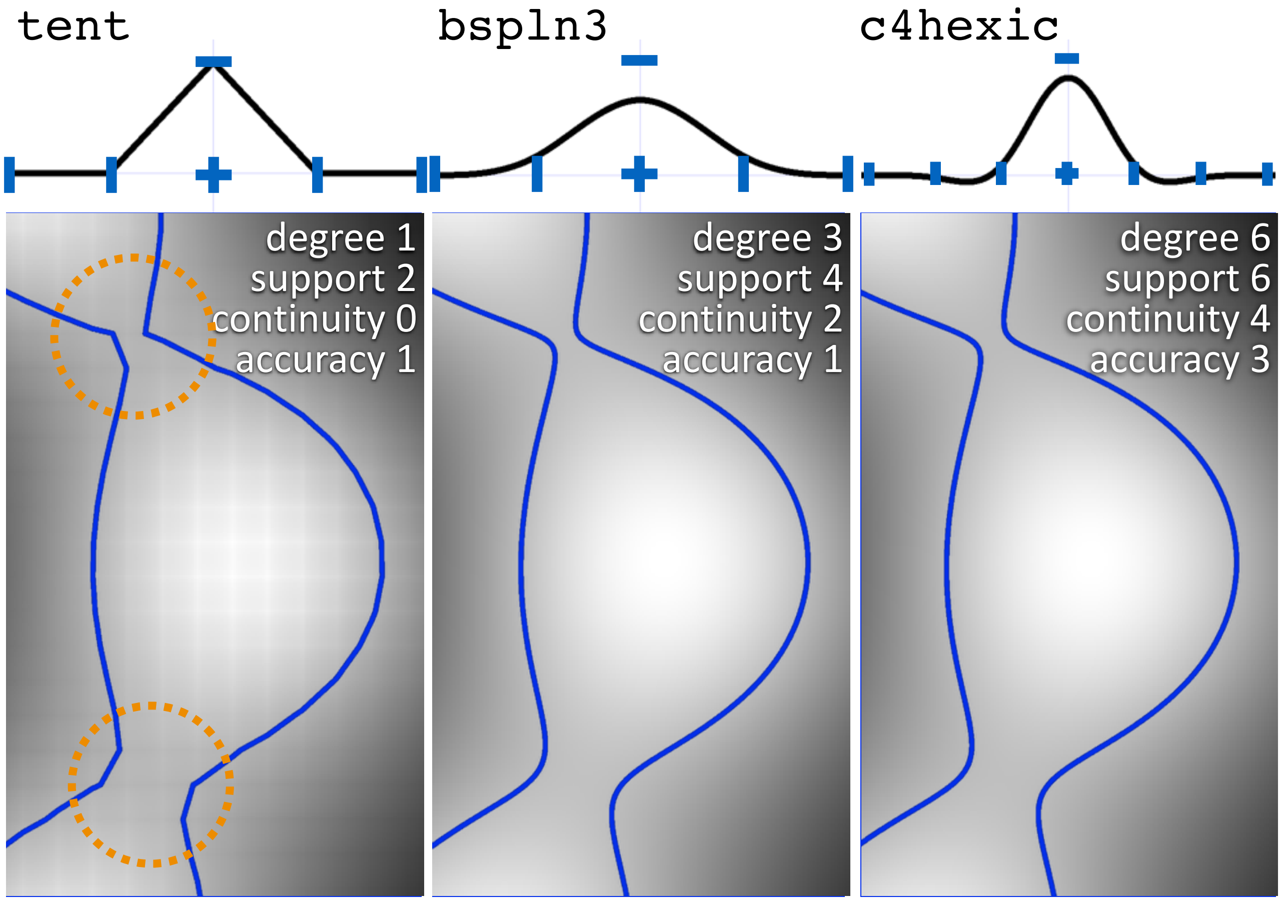
Results on more complex data

Conclusions

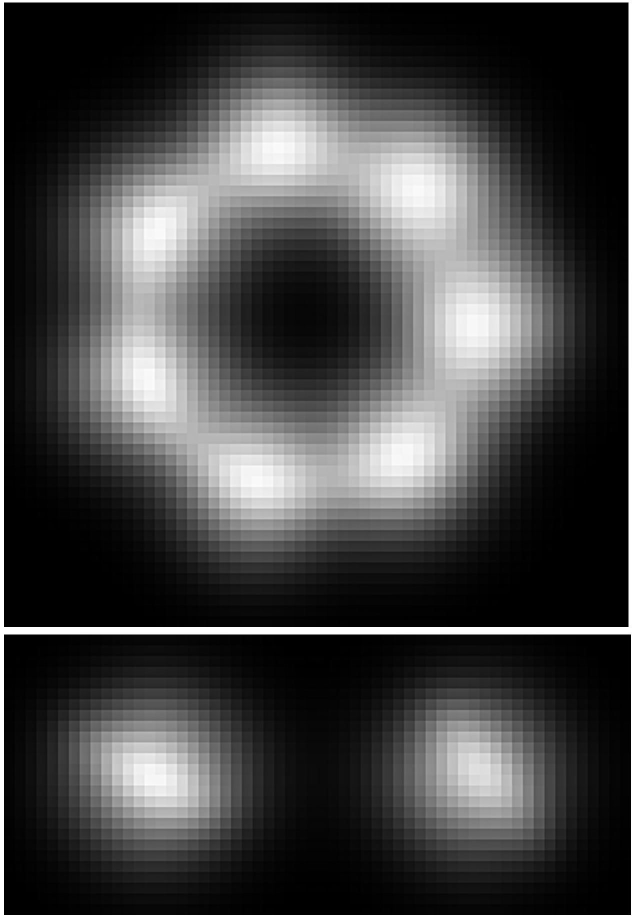
Example: isosurfaces on synthetic data

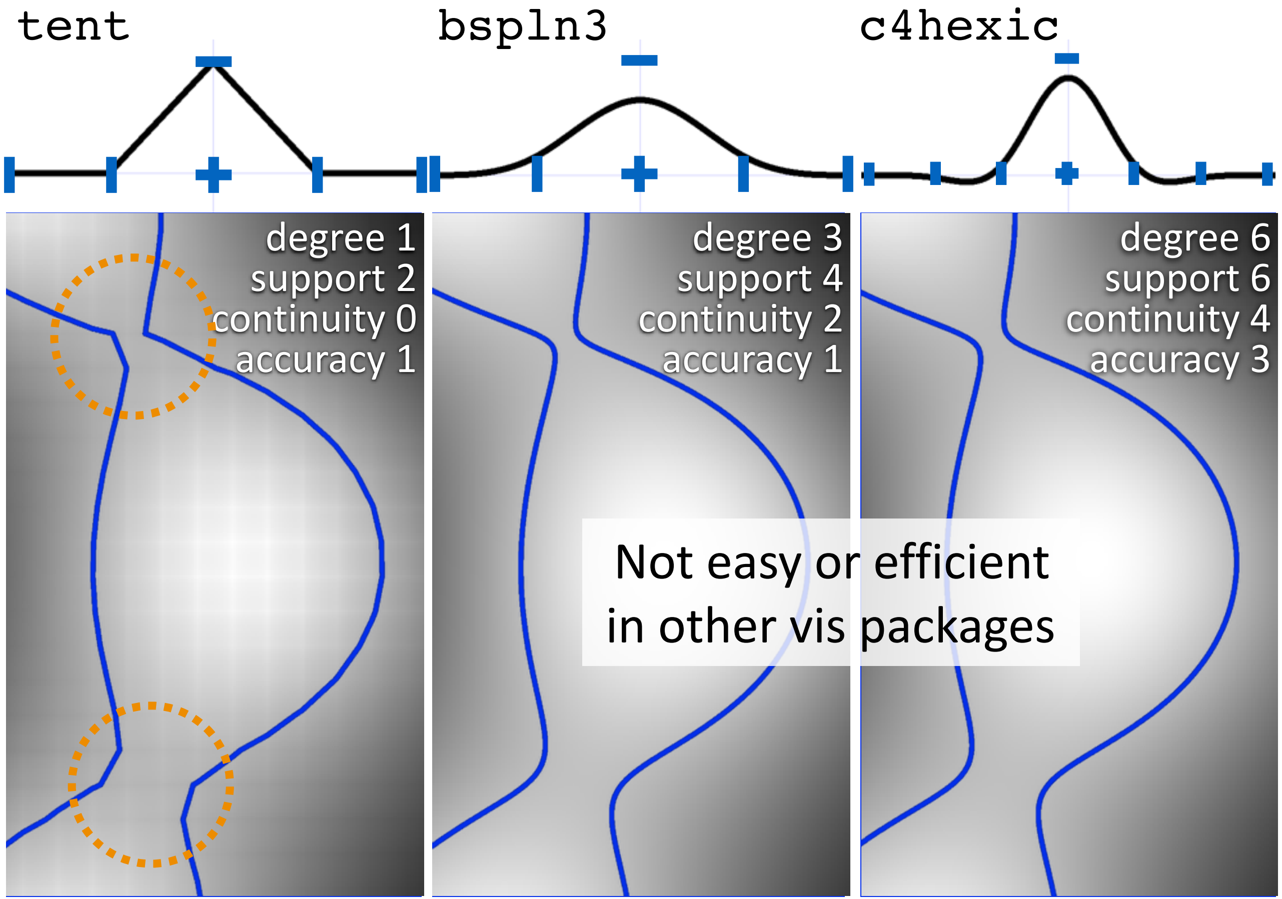






Example: isosurfaces on synthetic data





Not easy or efficient in other vis packages

(minimal) Volume Renderer in Diderot

```
1  input vec3 camEye ("Camera look-from point"); // look-at = [0,0,0]
2  input real camDepth ("Distance between near,far clip planes");
3  input real camFOV ("Vertical angle subtended by image");
4  input int imgRes ("Resolution on edge of square output image");
5  input real rayStep ("Sampling distance on central ray");
6  input real thick ("Apparent thickness of isosurface");
7  input real v0 ("which isosurface to render");
8  input image(3)[] vol ("data to render");
9  field#2(3)[] F = bspln3 ⊛ vol; // convolve image w/ recon kernel
10 // Only these feature functions are specific to isosurfaces
11 function vec3 fStep(vec3 x) = (v0 - F(x))*∇F(x)/(∇F(x)•∇F(x));
12 function real fStrength(vec3 x) = |∇F(x)|;
13 // Computing ray parameters and view-space basis
14 vec3 camN = normalize(-camEye);           // N: away from eye
15 vec3 camU = normalize(camN × [0,0,1]); // U: right
16 vec3 camV = camN × camU;                  // V: down
17 real camNear = |camEye| - camDepth/2;  // near clip, view space
18 real camFar  = |camEye| + camDepth/2;  // far clip, view space
19 // Core opacity function is a capped tent function
20 function real atent(real d) = clamp(0, 1, 1.5*(1 - |d|/thick));
21 // Renders ray through (rayU,rayV) on view plane through origin
22 strand ray(int ui, int vi) {
23   real UVmax = tan(camFOV*π/360)*|camEye|;
24   real rayU = lerp(-UVmax, UVmax, -0.5, ui, imgRes-0.5);
25   real rayV = lerp(-UVmax, UVmax, -0.5, vi, imgRes-0.5);
26   vec3 rayVec = camN + (rayU*camU + rayV*camV)/|camEye|;
27   real rayN = camNear - rayStep;    // init ray position
28   output vec4 rgba = [0,0,0,0];     // output ray color
29   real gray = 0;                    // ray grayscale
30   real tt = 1;                      // ray tranparency
31   update {
32     rayN += rayStep;        // increment ray position
33     if (rayN > camFar) {    // done if ray passed far plane
34       real q = 1-tt if tt < 1 else 1; // un-pre-multiply
35       rgba = [gray/q, gray/q, gray/q, 1-tt];
36       stabilize;
37     }
38     vec3 pos = camEye + rayN*rayVec;  // ray sample position
39     if (!inside(pos,F) || fStrength(pos) == 0) {
40       continue; // neither in field nor possibly near feature
41     }
42     vec3 step = fStep(pos);           // step towards feature
43     real aa = atent(|step|);          // sample opacity
44     if (aa == 0) { continue; }        // skip if no opacity
45     real gg = (normalize(step)•[0,0,1])^2; // 2-sided lighting
46     gray += tt*aa*((0.2 + 0.8*gg)); // ambient and diffuse
47     tt *= 1 - aa;                     // tranparencies multiply
48   }
49 } // end strand
50 initially [ray(ui,vi) | vi in 0..imgRes-1, ui in 0..imgRes-1];
```

Figure 2: A minimal but complete volume renderer is made specific to isosurfaces only by `fStep` and `fStrength` on lines 11 and 12.

(minimal) Volume Renderer in Diderot

```
1  input vec3 camEye ("Camera look-from point"); // look-at = [0,0,0]
2  input real camDepth ("Distance between near,far clip planes");
3  input real camFOV ("Vertical angle subtended by image");
4  input int imgRes ("Resolution on edge of square output image");
5  input real rayStep ("Sampling distance on central ray");
6  input real thick ("Apparent thickness of isosurface");
10 // Only these feature functions are specific to isosurfaces
14 vec3 camN = normalize(-camEye);               // N: away from eye
15 vec3 camU = normalize(camN × [0,0,1]); // U: right
16 vec3 camV = camN × camU;                      // V: down
17 real camNear = |camEye| - camDepth/2;  // near clip, view space
18 real camFar  = |camEye| + camDepth/2;  // far clip, view space
19 // Core opacity function is a capped tent function
20 function real atent(real d) = clamp(0, 1, 1.5*(1 - |d|/thick));
21 // Renders ray through (rayU,rayV) on view plane through origin
22 strand ray(int ui, int vi) {
23   real UVmax = tan(camFOV*π/360)*|camEye|;
24   real rayU = lerp(-UVmax, UVmax, -0.5, ui, imgRes-0.5);
25   real rayV = lerp(-UVmax, UVmax, -0.5, vi, imgRes-0.5);
26   vec3 rayVec = camN + (rayU*camU + rayV*camV)/|camEye|;
27   real rayN = camNear - rayStep;    // init ray position
28   output vec4 rgba = [0,0,0,0];     // output ray color
29   real gray = 0;                    // ray grayscale
30   real tt = 1;                      // ray tranparency
31   update {
35       rgba = [gray/q, gray/q, gray/q, 1-tt];
39     if (!inside(pos,F) || fSt
40       continue; // neither in
41     }
42     vec3 step = fStep(pos);
43     real aa = atent(|step|);
44     if (aa == 0) { continue; }
45     real gg = (normalize(step)•[0,0,1])^2; // 2-sided lighting
46     gray += tt*aa*((0.2 + 0.8*gg)); // ambient and diffuse
47     tt *= 1 - aa;                   // tranparencies multiply
48   }
49 } // end strand
50 initially [ray(ui,vi) | vi in 0..imgRes-1, ui in 0..imgRes-1];
```

```
field#2(3)[] F = bspln3 ⊛ vol; // convolve image w/ recon kernel
```

```
function vec3 fStep(vec3 x) = (v0 - F(x))*∇F(x)/(∇F(x)•∇F(x));
```

$$\epsilon = \frac{(v_0 - f(\mathbf{x}))\nabla f(\mathbf{x})}{\nabla f(\mathbf{x}) \cdot \nabla f(\mathbf{x})}$$

Figure 2: A minimal but complete volume renderer is made specific to isosurfaces only by `fStep` and `fStrength` on lines 11 and 12.

Diderot language supports: continuous fields, field operators like **∇**; enables mathematically idiomatic code

(minimal) Particle-Based Feature Sampler in Diderot

```
1  input real rad ("Inter-particle potential radius");
2  input real eps ("General convergence threshold");
3  input real v0 ("Which isosurface to sample");
4  input vec3{} ipos ("Initial point positions");
5  input image(3)[] vol ("Data to analyze");
6  field#2(3)[] F = bspln3 ⊛ clamp(vol); // convolve w/ recon kernel
7  // Only these three "f" functions are specific to isosurfaces
8  function vec3 fStep(vec3 x) = (v0 - F(x))*∇F(x)/(∇F(x)•∇F(x));
9  function tensor[3,3] fPerp(vec3 x) {
10   vec3 norm = normalize(∇F(x));
11   return identity[3] - norm⊗norm;
12 }
13 function real fStrength(vec3 x) = |∇F(x)|;
14 function real phi(real r) = (1 - r)^4;  // univariate potential
15 function real phi'(real r) = -4*(1 - r)^3;
16 function real enr(vec3 x) = phi(|x|/rad);
17 function vec3 frc(vec3 x) = phi'(|x|/rad) * (1/rad) * x/|x|;
18 // Strands first find feature, then interact w/ or make neighbors
19 strand point (vec3 pos0, real hh0) {
20   output vec3 pos = pos0;     // current particle position
21   real hh = hh0;              // energy gradient descent stepsize
22   vec3 step = [0,0,0];        // energy+feature steps this iter
23   bool found = false;         // whether feature has been found
24   int nfs = 0;                // number feature steps taken
25   update {
26     if (!inside(pos, F) || fStrength(pos) == 0) {
27       die; // not in field domain & not possibly near feature
28     }
29     if (!found) {                   // looking for feature
30       step = fStep(pos);            // one step towards feature
31       pos += step;
32       if (|step|/rad > eps) {       // took a substantial step
33         nfs += 1;
34         if (nfs > 10) { die; }      // too slow to converge
35       } else { found = true; }      // else converged on feature
36     } else {        // feature found; interact with other points
37       pos += fStep(pos);            // refine feature sampling
38       step = [0,0,0];               // initialize output step
39       real oldE = 0;                // energy at current location
40       vec3 force = [0,0,0];         // force on me from neighbors
41       int nn = 0;                   // number of neighbors
42       foreach (point P in sphere(rad)) {
43         oldE += enr(P.pos - pos);
44         force += frc(P.pos - pos);
45         nn += 1;
46       }
47       if (0 == nn) {                   // no neighbors, so create one
48         new point(pos + [0.5*rad,0,0], hh);
49         continue;
50       }                                // else interact w/ neighbors
51       force = fPerp(pos)•force;        // no force perp. to fStep(pos)
52       vec3 es = hh*force;              // energy step along force
53       if (|es| > rad) {                // limit motion to radius
54         hh *= rad/|es|;                // decrease stepsize and step
55         es *= rad/|es|;
56       }                                // now |es| <= rad
57       vec3 fs = fStep(pos+es);         // find step towards feature
58       if (|fs|/|es| > 0.5) {           // feature step too big
59         hh *= 0.5;                     // try again w/ smaller step
60         continue;
61       }
62       vec3 oldpos = pos;
63       pos += fs + es;                  // take steps, find new energy
64       real newE = sum { enr(pos - P.pos) | P in sphere(rad) };
65       if (newE - oldE > 0.5*(pos - oldpos)•(-force)) {
66         pos = oldpos;                  // energy didn't go down enough;
67         hh *= 0.5;                     // try again w/ smaller step
68         continue;
69       }
70       hh *= 1.1;                       // cautiously increase stepsize
71       step = fs + es;                  // record steps taken
72       if (nn < 5) {                    // add neighbor if have too few
73         new point(pos + 0.5*rad*normalize(es), hh);
74       }
75     } // else found
76   } // update
77 }
78 global {
79   bool allfound = all { P.found | P in point.all};
80   real maxstep = max { |P.step| | P in point.all };
81   if (allfound && maxstep/rad < eps) { stabilize; }
82 }
83 initially { point(ipos[ii], 1) | ii in 0 .. length(ipos)-1 };
```

Figure 4: A minimal but complete surface feature sampler is made specific to isosurfaces only by three feature functions starting line 8.

(minimal) Particle-Based Feature Sampler in Diderot

EuroVis 2018

```
1  input real rad ("Inter-particle potential radius");
2  input real eps ("General convergence threshold");
3  input real v0 ("Which isosurface to sample");
4  input vec3{} ipos ("Initial point positions");
5  input image(3)[] vol ("Data to analyze");
6  field#2(3)[] F = bspln3 ⊛ clamp(vol); // convolve w/ recon kernel
7  // Only these three "f" functions are specific to isosurfaces
8  function vec3 fStep(vec3 x) = (v0 - F(x))*∇F(x)/(∇F(x)•∇F(x));
9  function tensor[3,3] fPerp(vec3 x) {
10   vec3 norm = normalize(∇F(x));
11   return identity[3] - norm⊗norm;
12 }
13 function real fStrength(vec3 x) = |∇F(x)|;
14 function real phi(real r) = (1 - r)^4;  // univariate potential
15 function real phi'(real r) = -4*(1 - r)^3;
16 function real enr(vec3 x) = phi(|x|/rad);
17 function vec3 frc(vec3 x) = phi'(|x|/rad) * (1/rad) * x/|x|;
18 // Strands first find feature, then interact w/ or make neighbors
19 strand point (vec3 pos0, real hh0) {
20   output vec3 pos = pos0;     // current particle position
21   real hh = hh0;              // energy gradient descent stepsize
22   vec3 step = [0,0,0];        // energy+feature steps this iter
23   bool found = false;         // whether feature has been found
24   int nfs = 0;                // number feature steps taken
25   update {
26     if (!inside(pos, F) || fStrength(pos) == 0) {
27       die; // not in field domain & not possibly near feature
28     }
29     if (!found) {                  // looking for feature
30       step = fStep(pos);           // one step towards feature
31       pos += step;
32       if (|step|/rad > eps) {      // took a substantial step
33         nfs += 1;
34         if (nfs > 10) { die; }     // too slow to converge
35       } else { found = true; }     // else converged on feature
36     } else {       // feature found; interact with other points
37       pos += fStep(pos);           // refine feature sampling
38       step = [0,0,0];              // initialize output step
39       real oldE = 0;               // energy at current location
40       vec3 force = [0,0,0];        // force on me from neighbors
41       int nn = 0;                  // number of neighbors
42       foreach (point P in sphere(rad)) {
43         oldE += enr(P.pos - pos);
44         force += frc(P.pos - pos);
45         nn += 1;
46       }
47       if (0 == nn) {                   // no neighbors, so create one
48         new point(pos + [0.5*rad,0,0], hh);
49         continue;
50       }                                // else interact w/ neighbors
51       force = fPerp(pos)•force;        // no force perp. to fStep(pos)
52       vec3 es = hh*force;              // energy step along force
53       if (|es| > rad) {                // limit motion to radius
54         hh *= rad/|es|;                // decrease stepsize and step
55         es *= rad/|es|;
56       }                                // now |es| <= rad
57       vec3 fs = fStep(pos+es);         // find step towards feature
58       if (|fs|/|es| > 0.5) {           // feature step too big
59         hh *= 0.5;                     // try again w/ smaller step
60         continue;
61       }
62       vec3 oldpos = pos;
63       pos += fs + es;                  // take steps, find new energy
64       real newE = sum { enr(pos - P.pos) | P in sphere(rad) };
65       if (newE - oldE > 0.5*(pos - oldpos)•(-force)) {
66         pos = oldpos;                  // energy didn't go down enough;
67         hh *= 0.5;                     // try again w/ smaller step
68         continue;
69       }
70       hh *= 1.1;                       // cautiously increase stepsize
71       step = fs + es;                  // record step that
72       if (nn < 5) {                    // add neighbor if have too few
73         new point(pos + 0.5*rad*normalize(es), hh);
74       }
75     } // else found
76   } // update
77 }
78 global {
79   bool allfound = all { P.found | P in point.all};
80   real maxstep = max { |P.step| | P in point.all };
81   if (allfound && maxstep/rad < eps) { stabilize; }
82 }
83 initially { point(ipos[ii], 1) | ii in 0 .. length(ipos)-1 };
```

feature **strength**

Inter-particle dynamics

One iteration

Take steps until converged

Distribute myself w.r.t. neighbors …

… re-find feature

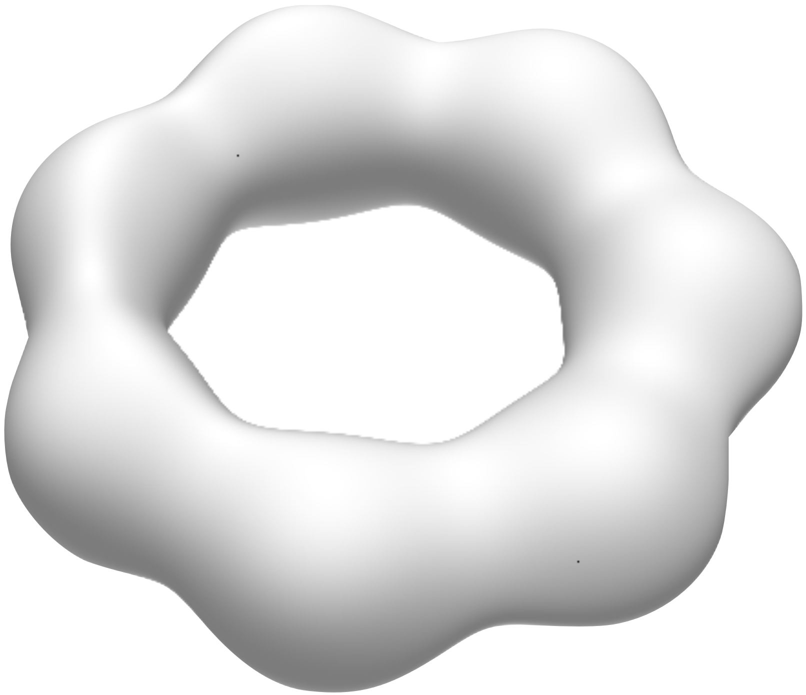
Ensure energy decrease

Control particle system population

Figure 4: A minimal but complete surface feature sampler is made specific to isosurfaces only by three feature functions starting line 8.

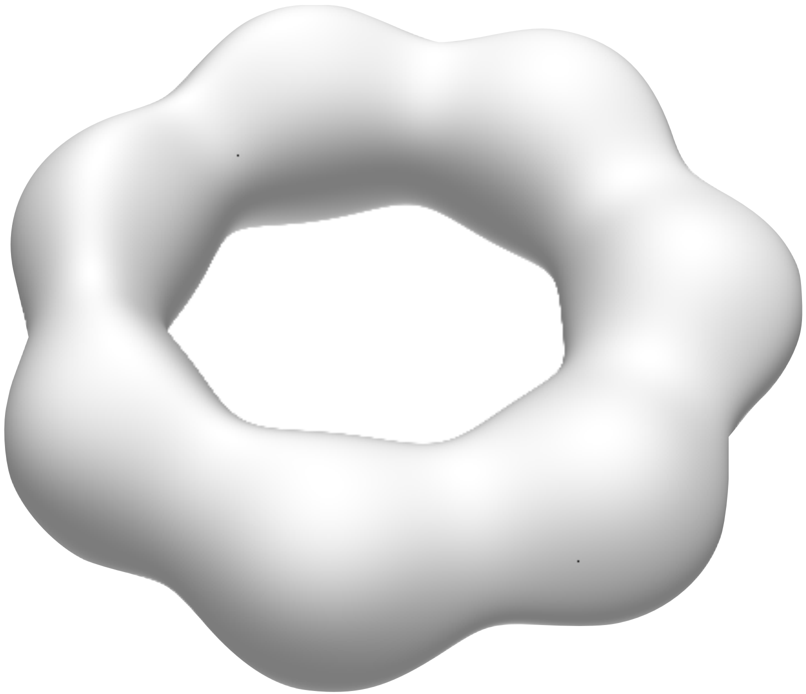
Rendering & sampling an isosurface

Direct Volume Rendering

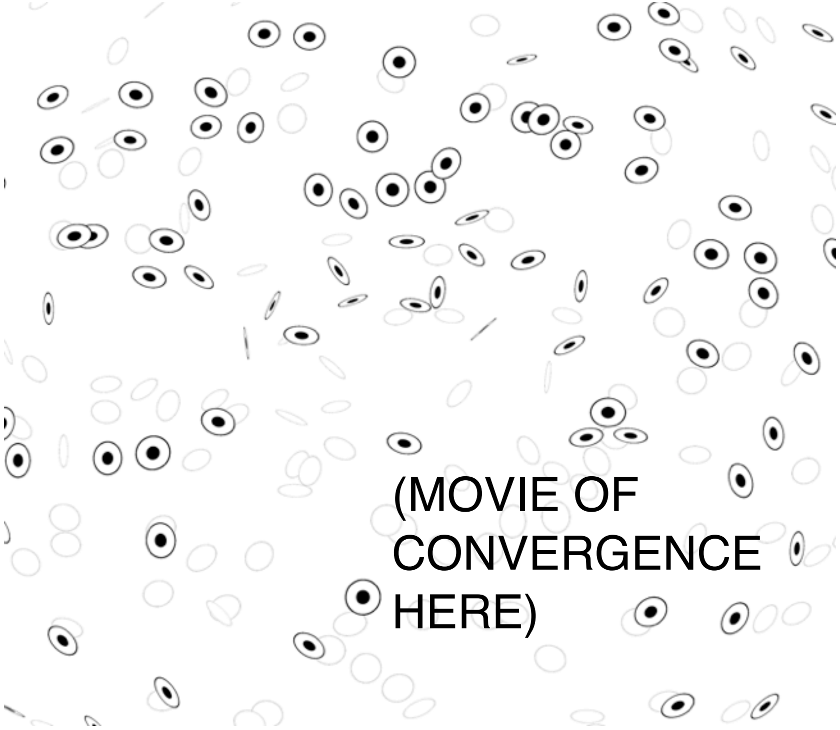


Rendering & sampling an isosurface

Direct Volume Rendering



Particle-based Feature Sampling



[Witkin-Particles-1994] [Crossno-Particles-1997] [Meyer-Robust-2005] …

Rendering & sampling **critical points**

Taylor series, for gradient:

$$\nabla f(\mathbf{x}+\boldsymbol{\epsilon}) \approx \nabla f(\mathbf{x}) + \mathbf{H}f(\mathbf{x})\boldsymbol{\epsilon}$$

Critical point:

$$\nabla f(\mathbf{x}) = 0$$

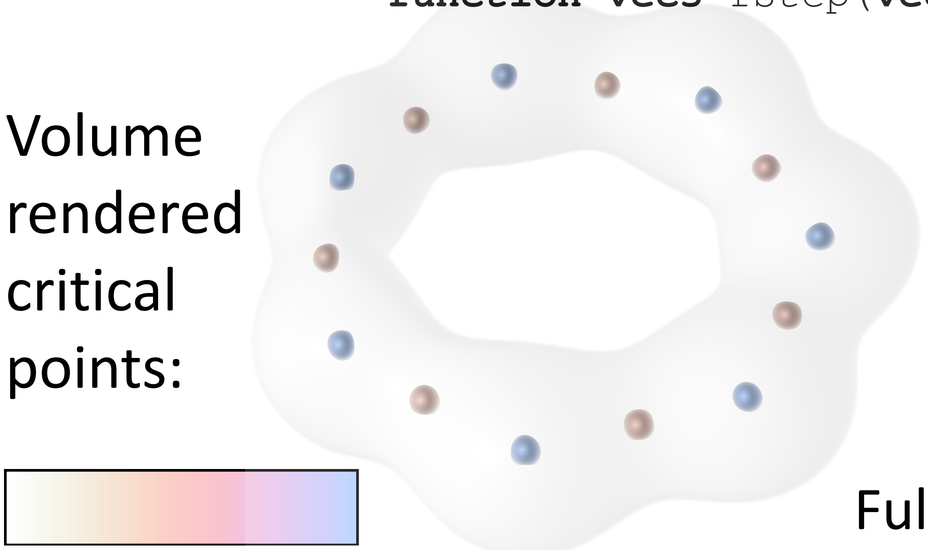
$$0 = \nabla f(\mathbf{x}) + \mathbf{H}f(\mathbf{x})\boldsymbol{\epsilon} \Rightarrow \boldsymbol{\epsilon} = -(\mathbf{H}f(\mathbf{x}))^{-1}\nabla f(\mathbf{x})$$

Newton optimization step

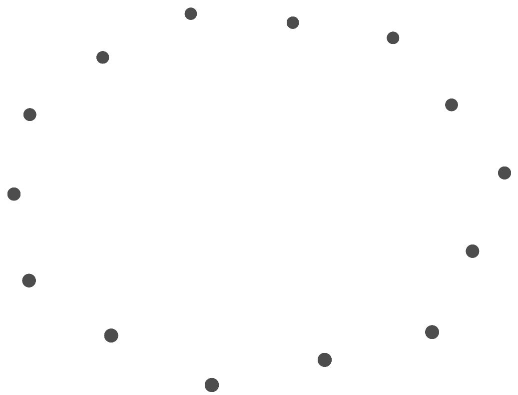
In Diderot:

```
function vec3 fStep(vec3 x) = -inv(∇⊗∇F(x))•∇F(x)
```

Volume rendered critical points:



Sampled critical points:



Full programs in Appendices A and B

Deriving Newton step for Ridges and Valleys

Eigensystem of Hessian: $\mathbf{H} = \sum_{i=0}^{2} \lambda_i \, \mathbf{e}_i \otimes \mathbf{e}_i \quad \lambda_0 \geq \lambda_1 \geq \lambda_2$

Ridge surface (rs) [Eberly-1996]: $\nabla f(\mathbf{x}) \cdot \mathbf{e}_2 = 0, \;\; \lambda_2 < 0$

Ridge line (rl): $\nabla f(\mathbf{x}) \cdot \mathbf{e}_2 = \nabla f(\mathbf{x}) \cdot \mathbf{e}_1 = 0, \;\; \lambda_1 < 0$

Projection onto relevant eigenvectors:

$$\mathbf{P}_{rs} = \mathbf{e}_2 \otimes \mathbf{e}_2 \quad \mathbf{P}_{rl} = \mathbf{e}_1 \otimes \mathbf{e}_1 + \mathbf{e}_2 \otimes \mathbf{e}_2$$
$$\mathbf{P}_{vs} = \mathbf{e}_0 \otimes \mathbf{e}_0 \quad \mathbf{P}_{vl} = \mathbf{e}_0 \otimes \mathbf{e}_0 + \mathbf{e}_1 \otimes \mathbf{e}_1$$

Necessary ridge,valley condition: $\mathbf{P} \, \nabla f = 0$

Newton step for Ridges and Valleys

Taylor series, for gradient:

$$\nabla f(\mathbf{x}+\boldsymbol{\epsilon}) \approx \nabla f(\mathbf{x}) + \mathbf{H}f(\mathbf{x})\boldsymbol{\epsilon}$$

$$\mathbf{P}\nabla f(\mathbf{x}+\boldsymbol{\epsilon}) \approx \mathbf{P}\nabla f(\mathbf{x}) + \mathbf{P}\mathbf{H}f(\mathbf{x})\boldsymbol{\epsilon}$$

Ridge,valley: $\mathbf{P}\,\nabla f = 0$

$$0 = \mathbf{P}\nabla f(\mathbf{x}) + \mathbf{P}\mathbf{H}f(\mathbf{x})\boldsymbol{\epsilon}$$

$\mathbf{P}$, $\mathbf{H}$ diagonal in $\{\mathbf{e}_i\}$ basis

$$\mathbf{H}f(\mathbf{x})\mathbf{P}\boldsymbol{\epsilon}$$

pick $\boldsymbol{\epsilon}$ in span of relevant $\mathbf{e}_i$

$$\mathbf{H}f(\mathbf{x})\boldsymbol{\epsilon}$$

$$\Rightarrow \boldsymbol{\epsilon} = -(\mathbf{H}f(\mathbf{x}))^{-1}\mathbf{P}\nabla f(\mathbf{x})$$

$\mathbf{P}$, $\mathbf{H}^{-1}$ diagonal in $\{\mathbf{e}_i\}$ basis

$$\Rightarrow \boldsymbol{\epsilon}_{rs} = -\frac{\mathbf{e}_2(\mathbf{e}_2 \cdot \nabla f(\mathbf{x}))}{\lambda_2}$$

Newton step for e.g. ridge surfaces

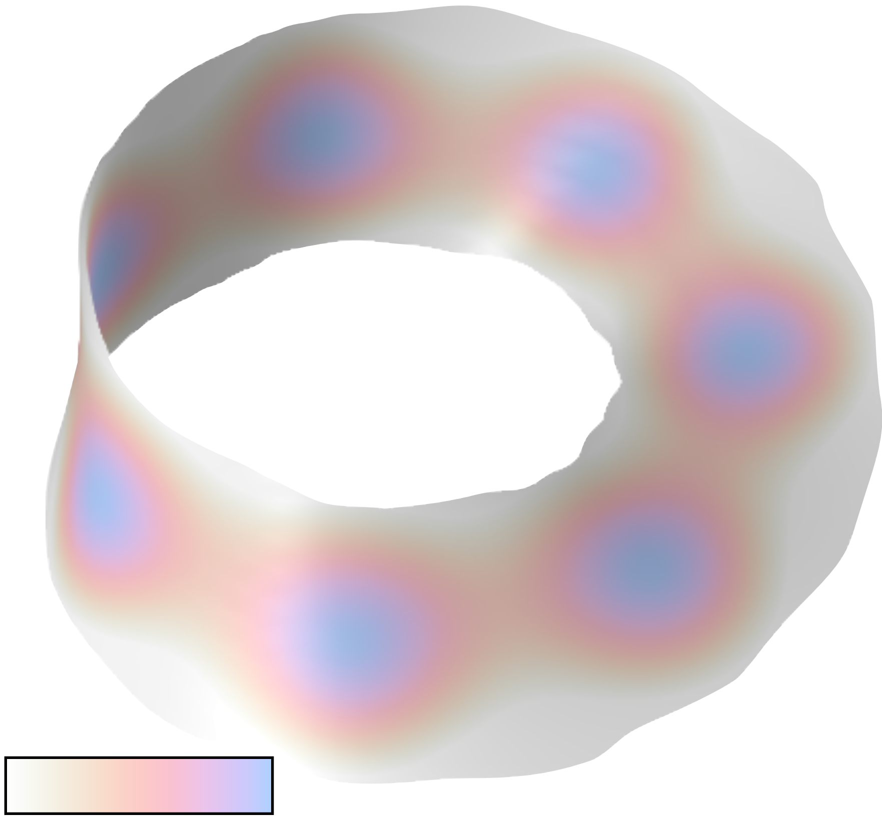
[Obermaier-MeshFree-2012]

In Diderot:

```
function vec3 fStep(vec3 x) {
  vec3{3} E = evecs(∇⊗∇F(x));
  real{3} L = evals(∇⊗∇F(x));
  return -(1/L{2})*E{2}⊗E{2}•∇F(x);
}
```

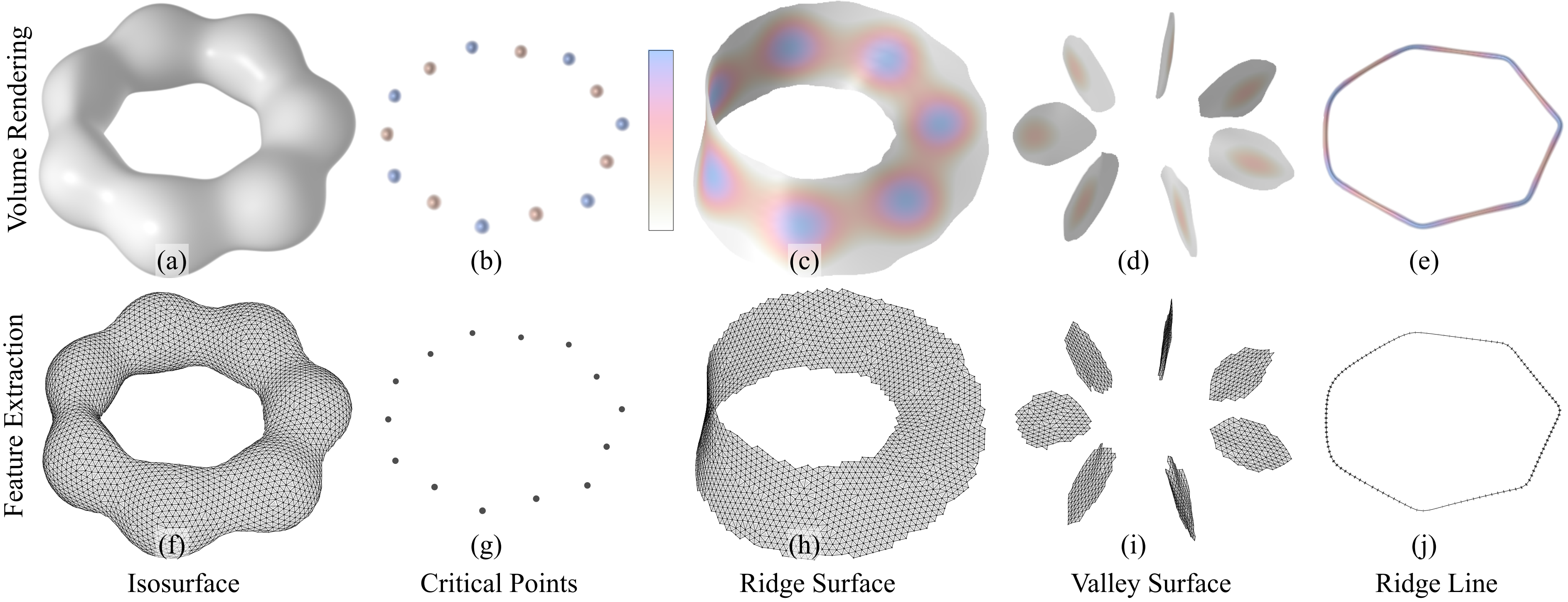
Rendering & sampling **ridge surfaces**

Direct Volume Rendering

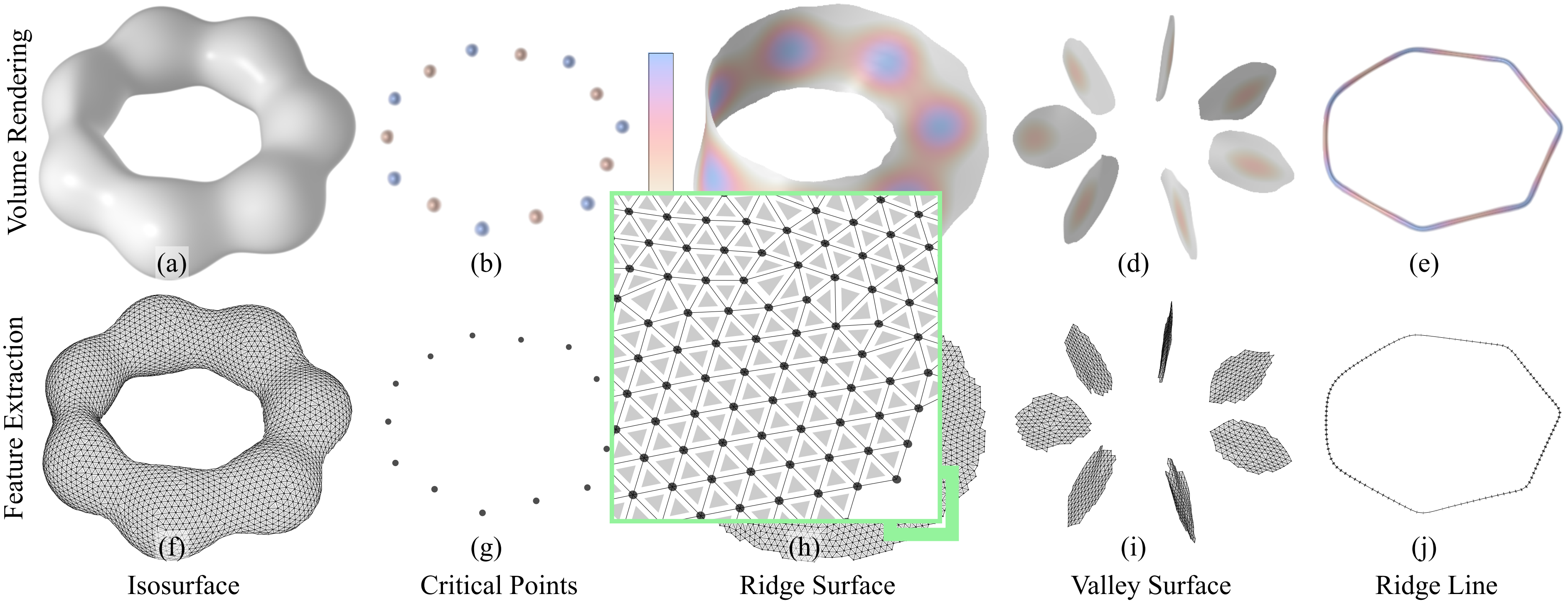


Particle-based Feature Sampling

Teaser Figure



Teaser Figure



Outline

Basic idea

Synthetic data: 2 vis methods, various features

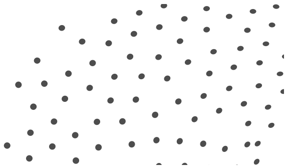
Technical aspects

- point meshing, feature strength

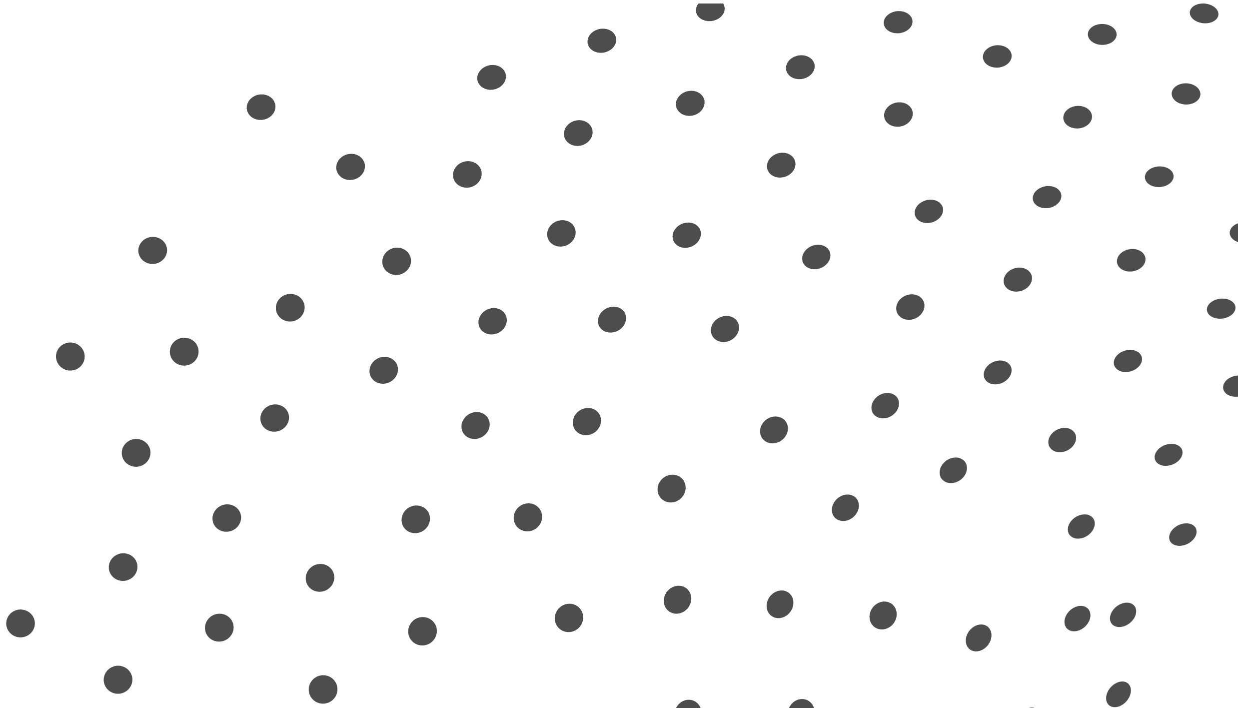
Results on more complex data

Conclusions

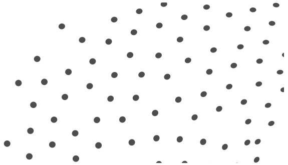
Meshing surface feature sampling

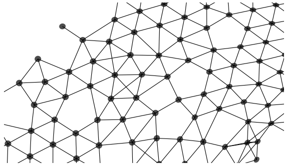


0) Points from converged particle system



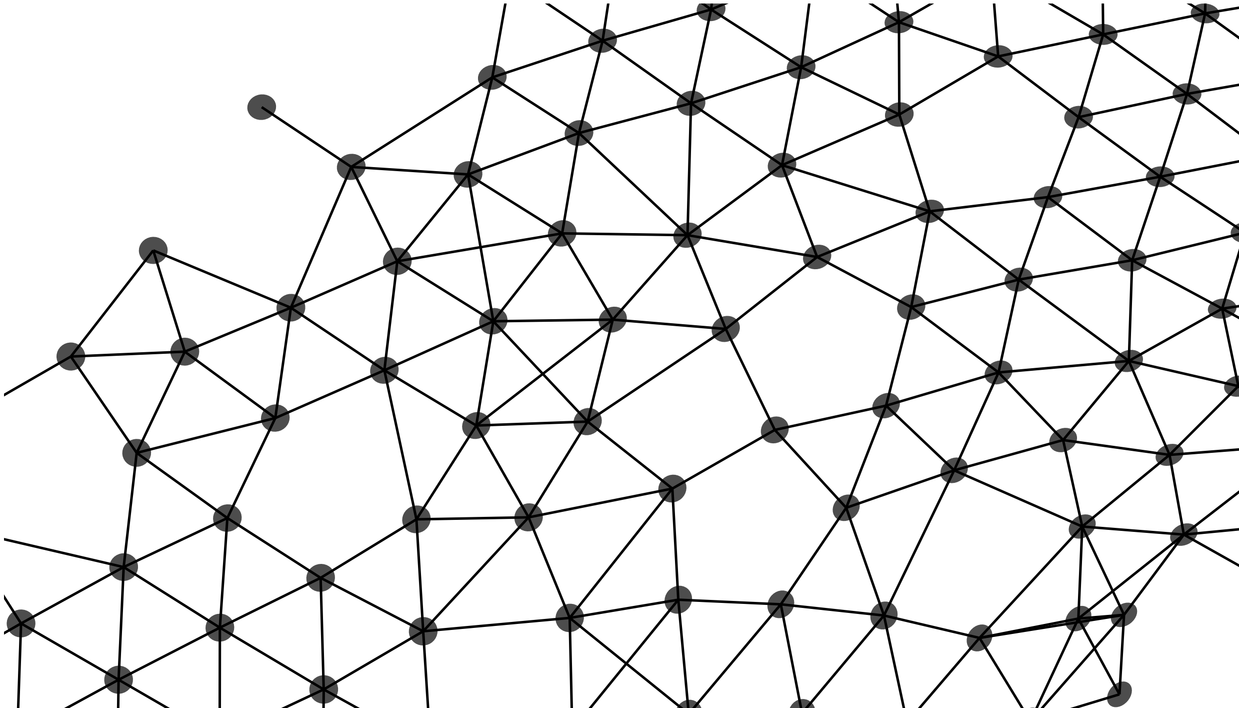
Meshing surface feature sampling



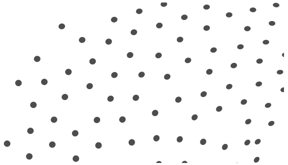


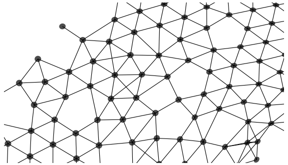
0) Points from converged particle system

1) Edges for all interacting particles



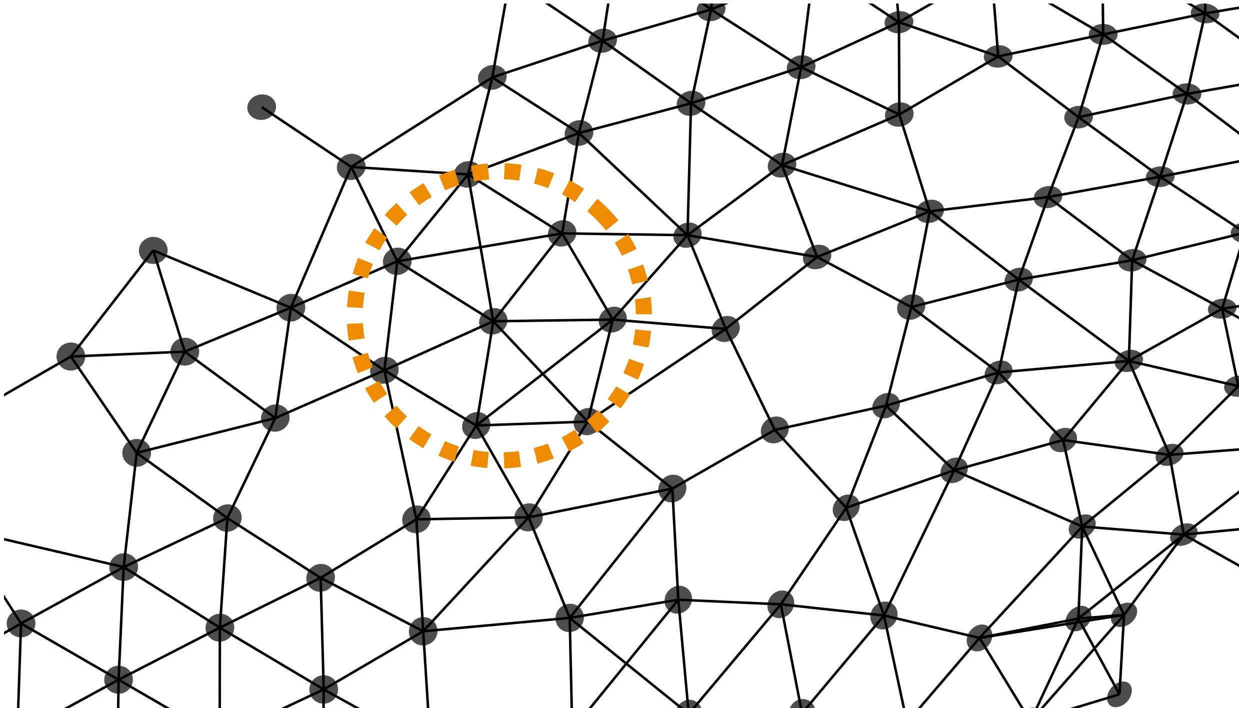
Meshing surface feature sampling



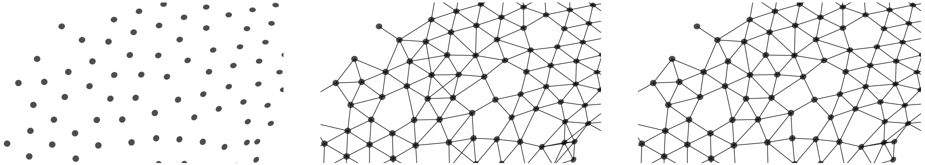


0) Points from converged particle system

1) Edges for all interacting particles



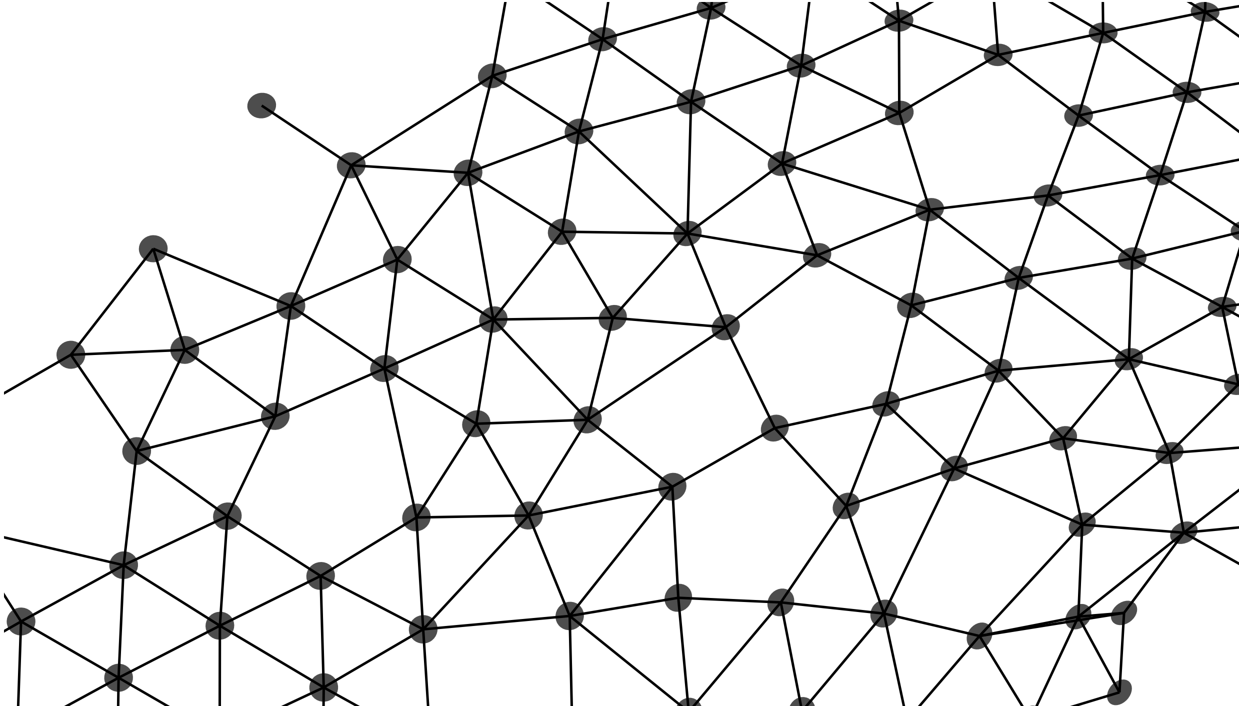
Meshing surface feature sampling



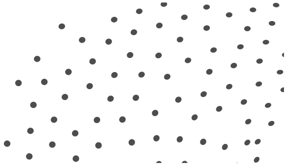
0) Points from converged particle system

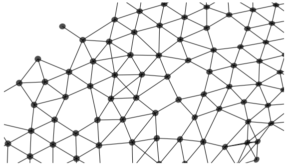
1) Edges for all interacting particles

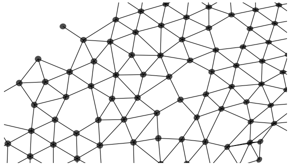
2) Removing crossed edges

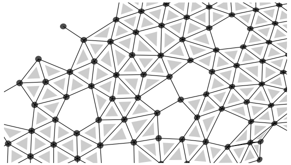


Meshing surface feature sampling







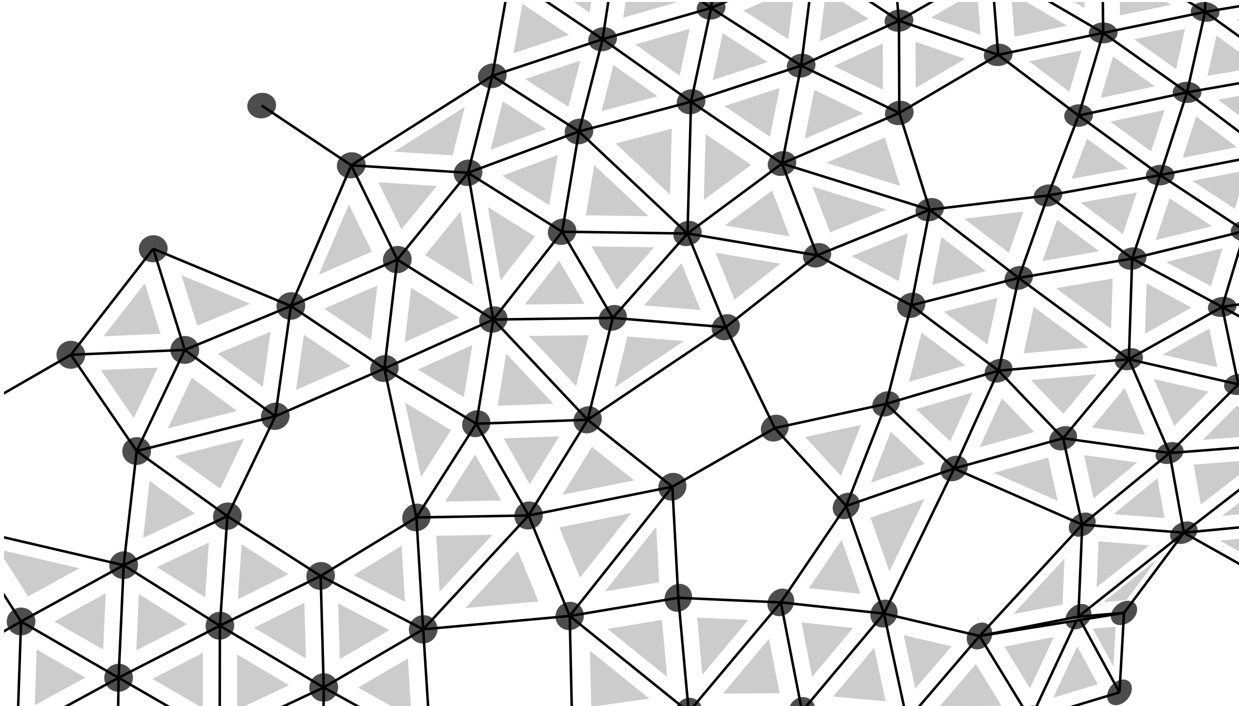


0) Points from converged particle system

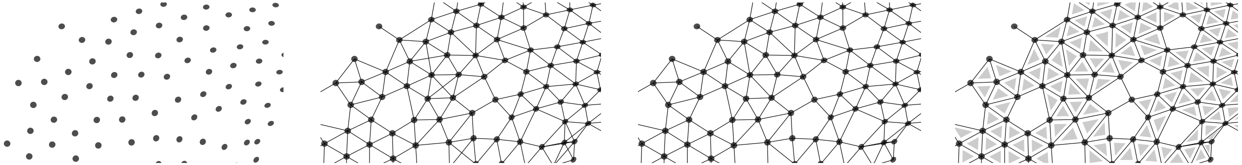
1) Edges for all interacting particles

2) Removing crossed edges

3) Triangulate 3-loops



Meshing surface feature sampling

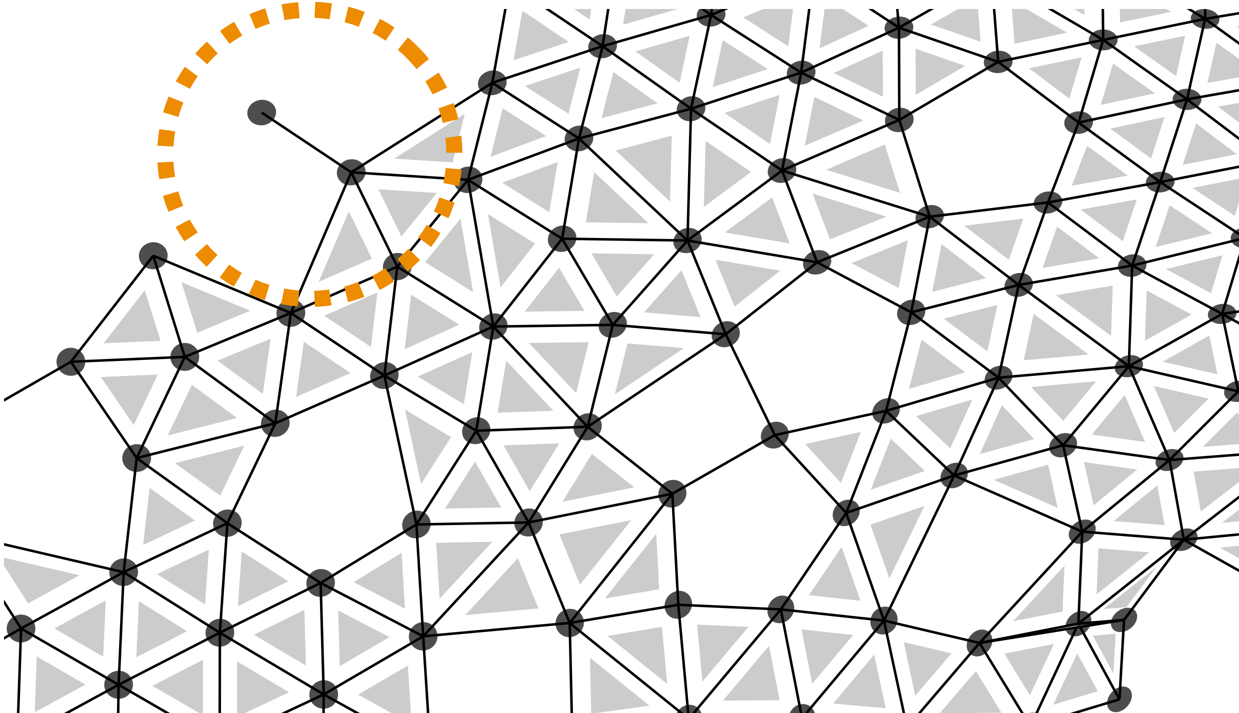


0) Points from converged particle system

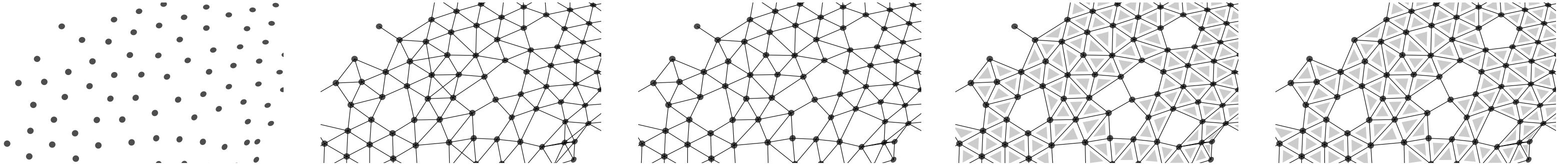
1) Edges for all interacting particles

2) Removing crossed edges

3) Triangulate 3-loops



Meshing surface feature sampling



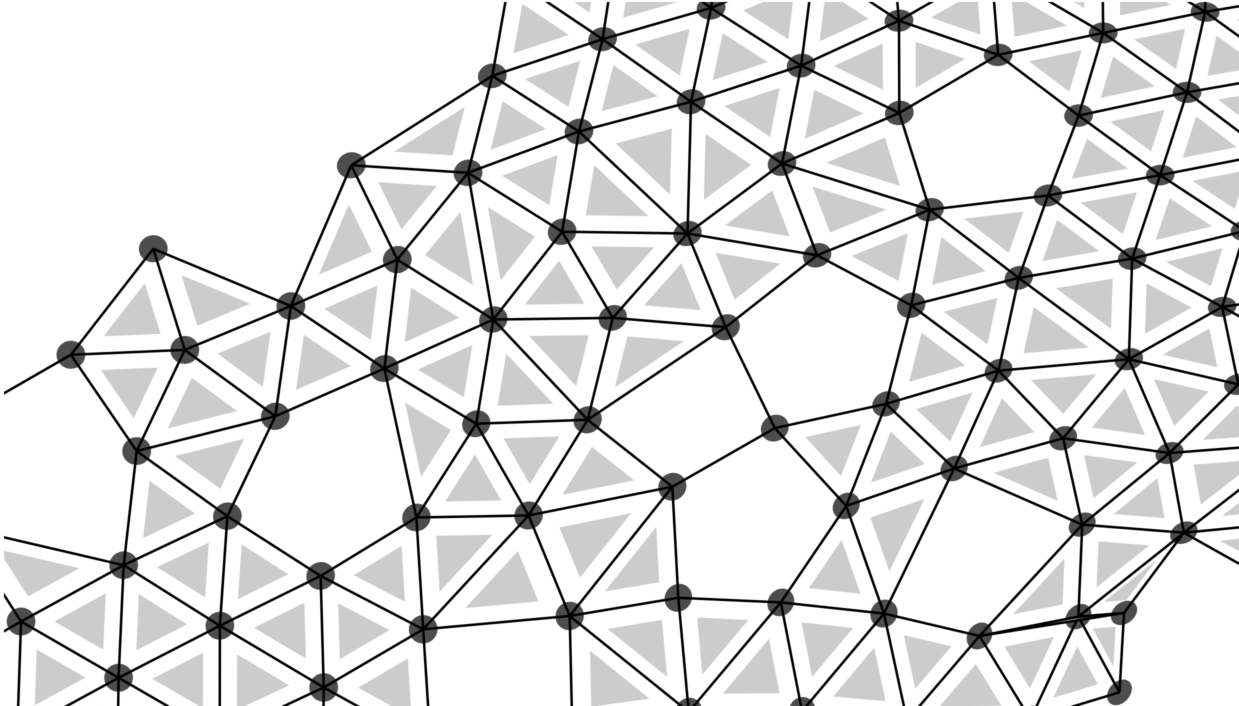
0) Points from converged particle system

1) Edges for all interacting particles

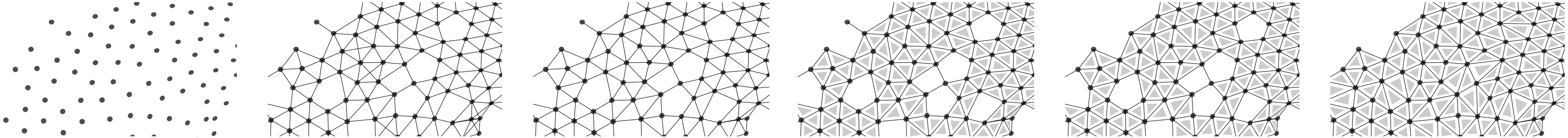
2) Removing crossed edges

3) Triangulate 3-loops

4) Remove stray edges



Meshing surface feature sampling



0) Points from converged particle system

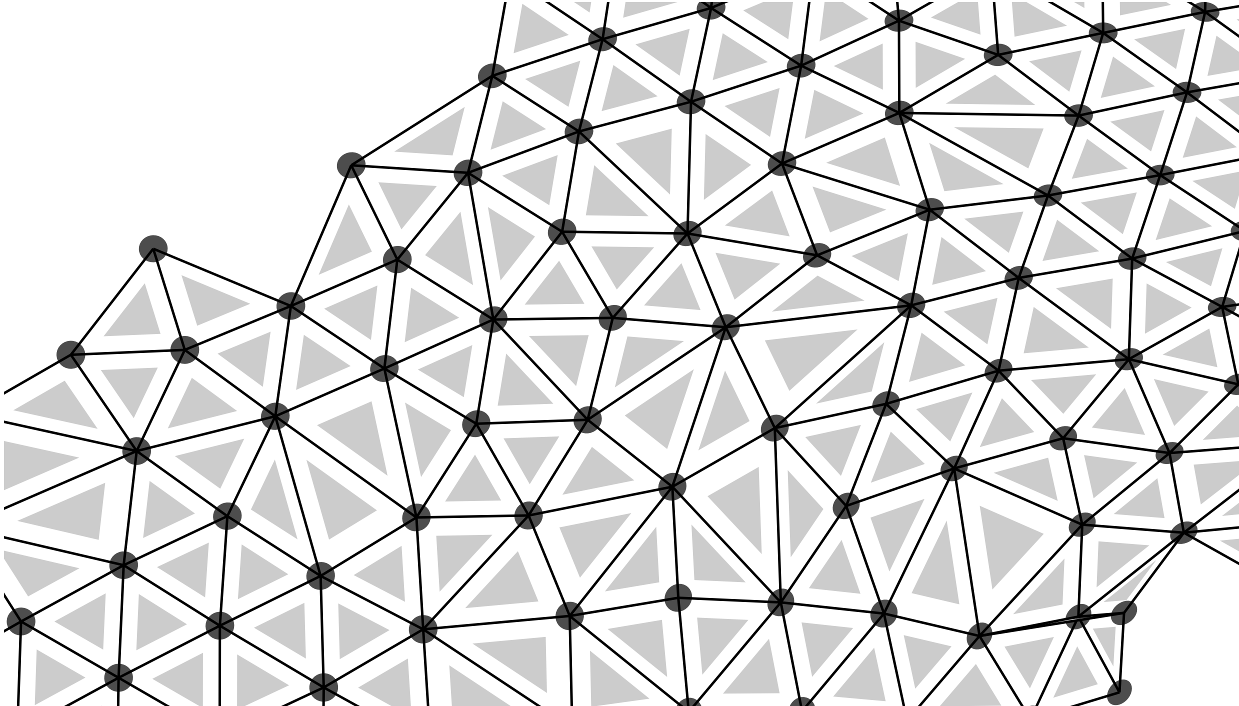
1) Edges for all interacting particles

2) Removing crossed edges

3) Triangulate 3-loops

4) Remove stray edges

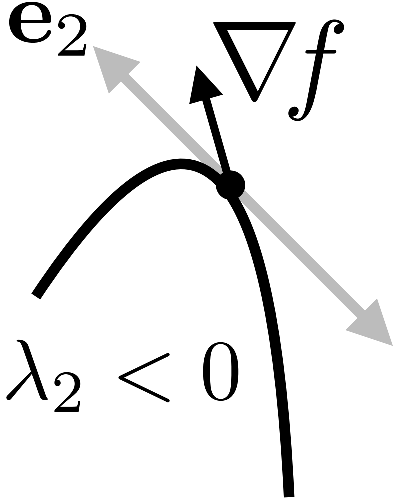
5) Triangulate holes



Feature strength (e.g. ridge surface)

Answers: when is it ok to skip looking for feature?

- Want only “real” features, not noise/artifact
- Unfortunately, depends on user-specified threshold T



$$\lambda_2 < T$$

$$T < 0$$

[Kindlmann-SSP-2009]

$$\frac{|\nabla f|}{\lambda_2} > T$$

$$T < 0$$

[Haralick-Ridges-1983]
[Schultz-CreaseSurf-2009]

We propose:

$$\frac{-\lambda_2}{r_0 + |\nabla f|} > T$$

$$r_0 > 0,\ T > 0$$

r_0 = strength bias

(analogous measures for other extremal features)

Outline

Basic idea

Synthetic data: 2 vis methods, various features

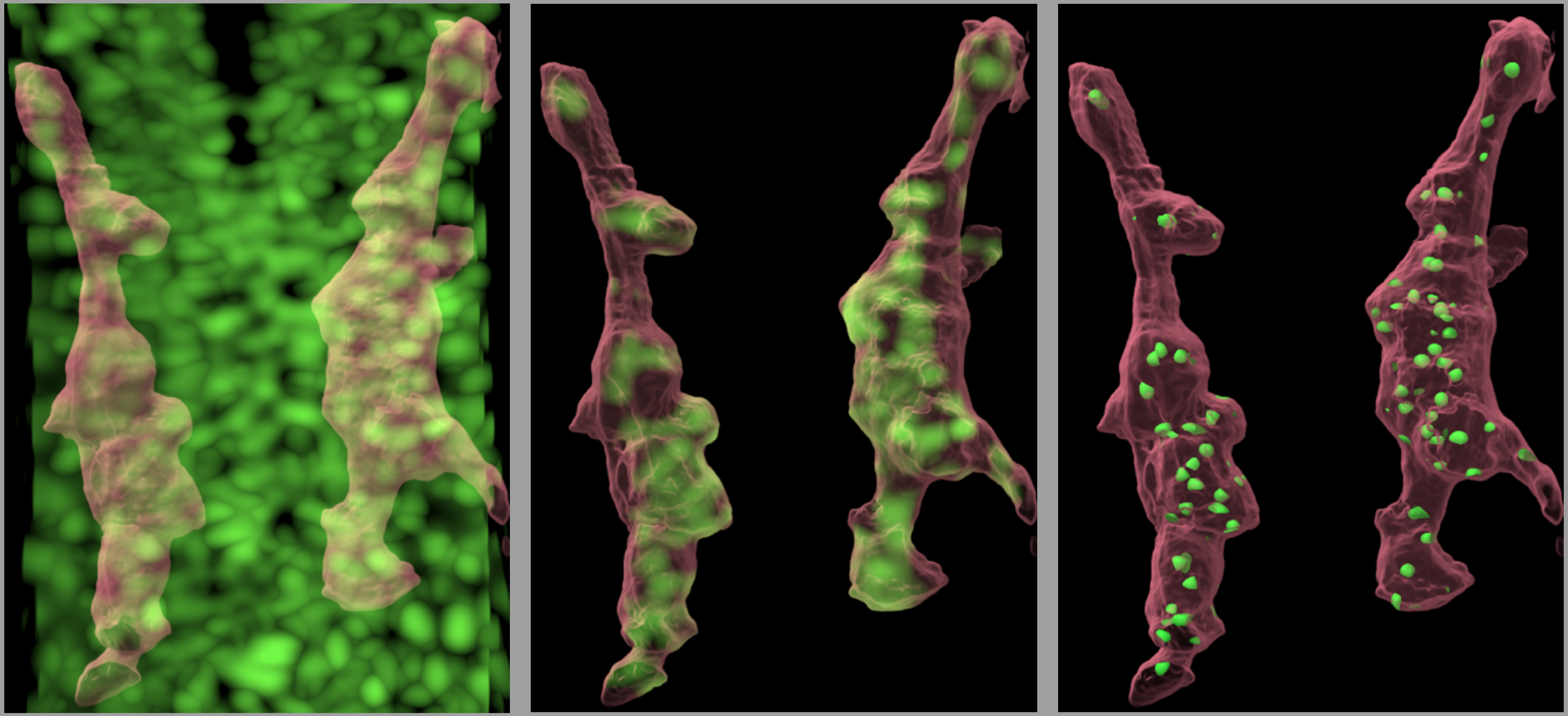
Technical aspects

point meshing, feature strength

Results on more complex data

Conclusions

Results: rendering local maxima



GFP: MIP & RFP: Iso

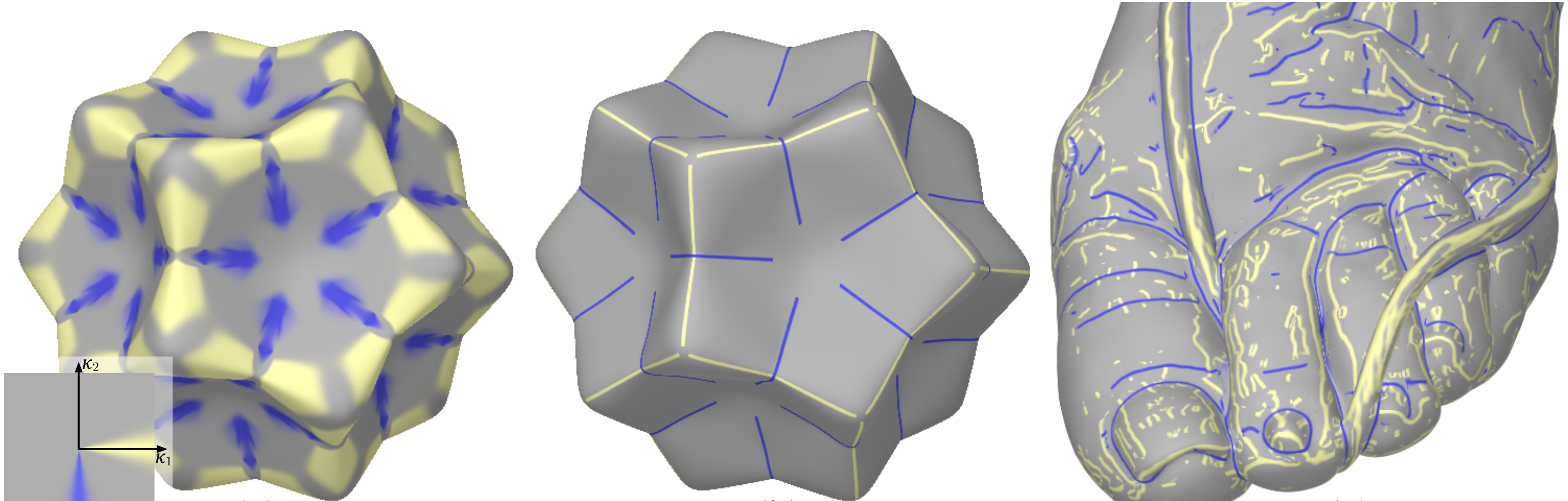
GFP: MIP inside RFP Iso

GFP maxima inside RFP Iso

Results: Rendering surface creases

Surface creases: points where curvature κ_i is extremal w.r.t motion along curvature direction $\boldsymbol{d}_i$ [Monga-CrestLines-1994]

```
field#3(3)[3] N = -∇F/|∇F|;
field#3(3)[3,3] P = identity[3] - N⊗N;
field#2(3)[3,3] G = P•(∇⊗N)•P;
field#2(3)[] discrim = sqrt(2*G:G - trace(G)*trace(G));
field#2(3)[] K1 = (trace(G) + discrim)/2;
field#2(3)[] K2 = (trace(G) - discrim)/2;
vec3 d1 = col1span(G(pos) - K2(pos)*P(pos));
real k1′ = d1•∇K1(pos);
real k1′′ = d1•(∇ ⊗ ∇K1(pos))•d1;
mcol = lerp(gray, rcol, atent(1,|k1′/k1′′|/3));
```



(a) (b) (c)

Results: Sampling vortex cores (vector data)

"Q" invariant of Jacobian [Chong-3DFlow-1990]:

where e is the eigenvector. The eigenvalues can be determined by solving the characteristic equation

$$\det[\mathbf{A} - \lambda I] = 0, \qquad (3)$$

which, for a 3×3 matrix, can be written as

$$\lambda^3 + P\lambda^2 + Q\lambda + R = 0, \qquad (4)$$

where

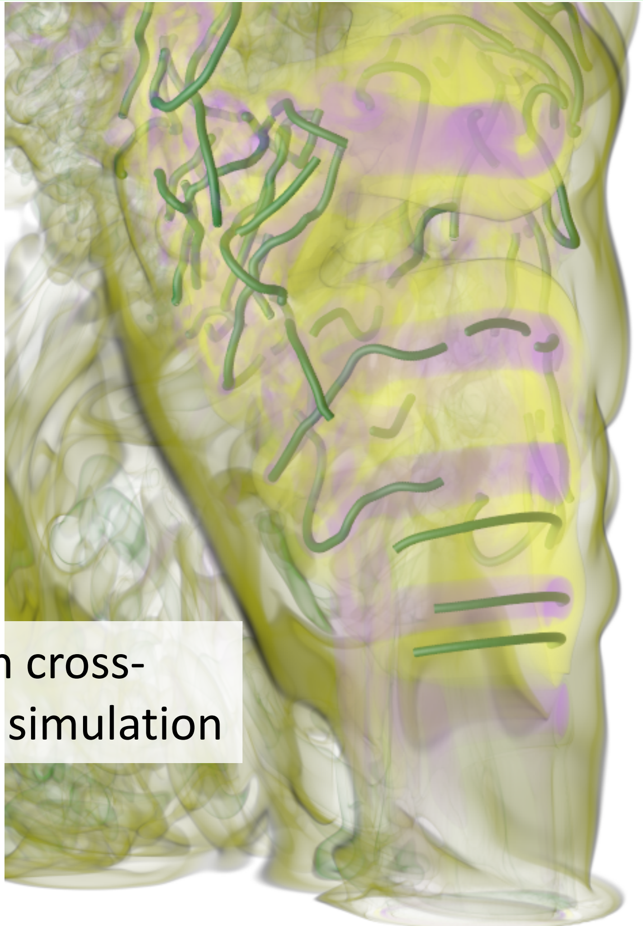
$$P = -(a_{11} + a_{22} + a_{33}) = -\mathrm{tr}[\mathbf{A}] = -S_{ii}, \qquad (5)$$

$$Q = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} = \tfrac{1}{2}(P^2 - \mathrm{tr}[\mathbf{A}^2]) = \tfrac{1}{2}(P^2 - S_{ij}S_{ji} - R_{ij}R_{ji}), \qquad (6)$$

"Q-criterion" [Hunt-Eddies-1988] suggests finding **ridge lines of Q**

Diderot code:

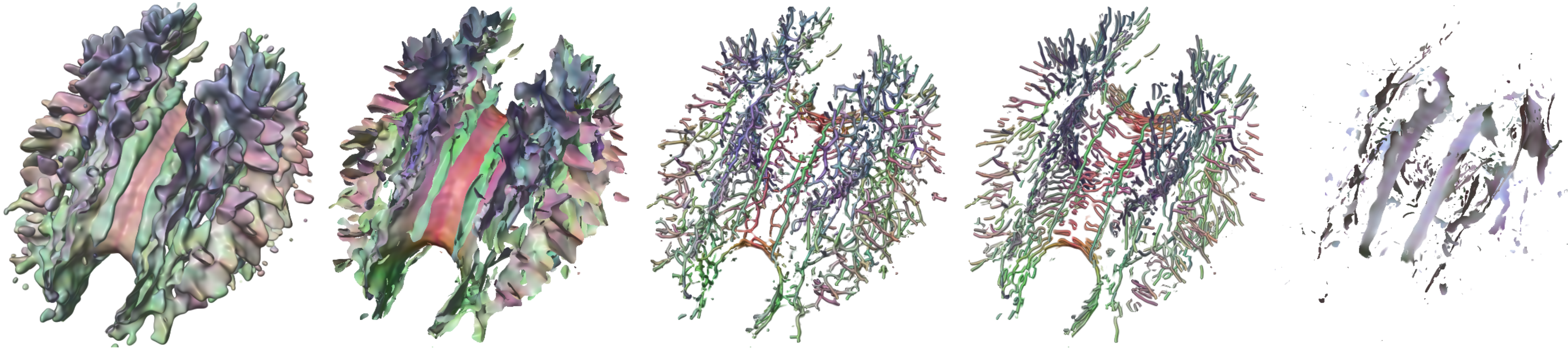
```
field#4[] V = c4hexic ⊛ img;
field#3(3)[3,3] J = ∇⊗V;
field#3(3)[] Q = (trace(J)^2 - trace(J•J))/2;
```



Jet in cross-flow simulation

Results: Anisotropy features (tensor data)

EuroVis 2018



(a) Isosurface (b) Ridge Surface (c) Ridge Line (d) Max Line (e) Min Surface

Figure 10: Different feature step functions reveal a variety of Fractional Anisotropy (FA) features from a DTI volume.

Outline

Basic idea

Synthetic data: 2 vis methods, various features

Technical aspects

  point meshing, feature strength

Results on more complex data

Conclusions

Conclusions: contributions

Advocating principled approach to vis implementation

Vis method and target feature connected by Newton step

Diderot enabled this exploratory research:

easy to express Newton steps for a variety of features

implement volume renderer & particle-based feature sampler

Meshing particles sampling (non-isosurface) surface features

Reproducibility: all Diderot code used is in paper and its appendices

Conclusions: future work

Accelerate particle system (execution time = minutes)
  (numerical methods)

Verify if particle system convergence justifies meshing strategy
  (computational geometry)

Curvature-dependent particle sampling

Build a GUI, integrate with other vis tools

References (in order of appearance)

EuroVis 2018

[Levoy-CGnA-1988] *Display of Surfaces from Volume Data*. M Levoy. IEEE Computer Graphics & Applications, 8(5):29–37, 1988

[Guy-PAMI-1997] *Inference of surfaces, 3D curves, and junctions from sparse, noisy, 3D data*. G Guy and G Medioni. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(11):1265–1277, November 1997.

[Tang-VIS-1998] *Extremal Feature Extraction from 3-D Vector and Noisy Scalar Fields*. CK Tang and G Medioni. In Proc. IEEE Visualization '98, pages 95–102, October 1998

[Amenta-SIGGRAPH-2004] *Defining point-set surfaces*. N Amenta and YJ Kil. Computer Graphics (Proc. SIGGRAPH), pages 264–270, 2004

[Witkin-Particles-1994] *Using Particles to Sample and Control Implicit Surfaces*. AP Witkin and PS Heckbert. Computer Graphics (Proc. SIGGRAPH), 28:269–277, 1994

[Crossno-Particles-1997] *Isosurface Extraction Using Particle Systems*. P Crossno and E Angel. In Proc. IEEE Vis, pages 495–498, 1997

[Meyer-Robust-2005] *Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces*. MD Meyer, P Georgel, and RT Whitaker. In Proc. Shape Modeling and Applications (SMI), pages 124–133, June 2005

[Eberly-1996] *Ridges in Image and Data Analysis*. D Eberly. Kluwer Academic Publishers, 1996

[Obermaier-MeshFree-2012] *On mesh-free valley surface extraction with application to low frequency sound simulation*. H Obermaier, J Mohring, E Deines, M Hering-Bertram, and H Hagen. IEEE TVCG, 18(2):270–282, 2012.

[Kindlmann-SSP-2009] *Sampling and Visualizing Creases with Scale-Space Particles. GL Kindlmann, RSJ Estépar, SM Smith, and C-F Westin*. IEEE TVCG, 15(6):1415–1424, 2009

[Haralick-Ridges-1983] *Ridges and Valleys on Digital Images*. RM Haralick. Computer Vision, Graphics, and Image Processing, 22:28–38, 1983

[Monga-CrestLines-1994] *Crest lines extraction in volume 3D medical images: a multi-scale approach*. O Monga, R Lengagne, and R Deriche. In Proceedings of 12th International Conference on Pattern Recognition, volume 1, pages 553–555 vol.1, October 1994

[Chong-3DFlow-1990] *A general classification of three-dimensional flow fields. MS Chong, AE Perry, and BJ Cantwell*. Physics of Fluids A: Fluid Dynamics, 2(5):765–777, 1990

[Hunt-Eddies-1988] *Eddies, stream, and convergence zones in turbulent flows*. JCR Hunt, A Wray, and P Moin. In Proceedings of the Summer Program, pages 193–208. Center for Turbulence Research (Stanford University), 1988

Thank you

Biology collaborators: Ana Beiriger and Prof. Victoria Prince

NSF grants CCF-1564298 and IOS-1555972

Questions?

Learn from Diderot examples (google: diderot examples)

https://github.com/Diderot-Language/examples

Reproducible Diderot figures (google: reproducible diderot)

https://github.com/Diderot-Language/reproduce

Google group: https://groups.google.com/forum/#!forum/diderot-language

Email glk@uchicago.edu to join **slack** channel