Diffusion Generated Motion in Vision Applications

Selim Esedoglu

Threshold Dynamics

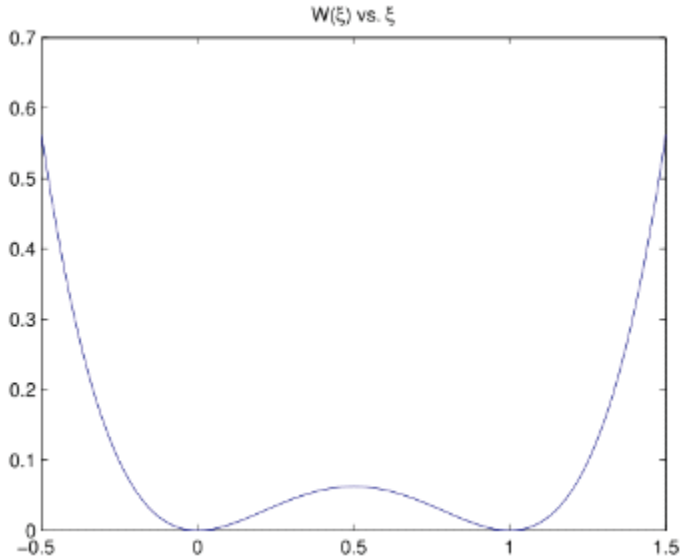
- Merriman-Bence-Osher (1992)
- To address stiffness of Allen-Cahn, time split the equation:
  - Step 1:

    $$u_t = \Delta u$$

  - Step 2:

    $$u_t = -\frac{1}{\varepsilon^2} W'(u)$$

- When $t \to \infty$, Step 2 turns into thresholding:



$$\lim_{t\to\infty} u(x,t) = \begin{cases} 0 & \text{if } u(x,0) < \frac{1}{2}, \\ 1 & \text{if } u(x,0) > \frac{1}{2}. \end{cases}$$

Threshold Dynamics

Fix a time step size $\delta t > 0$ and generate a discrete in time approximation $\{\Sigma^k\}_{k=0}^{\infty}$ to the flow as follows:

- MBO Algorithm:

  1. Convolution Step:

$$u(x,t) = \mathbf{1}_{\Sigma^k} * G_{\delta t} \quad \text{where} \quad G_t(x) = \frac{1}{(4\pi t)^{\frac{n}{2}}} e^{-\frac{|x|^2}{4t}}$$

  2. Thresholding Step:

$$\Sigma^{k+1} = \left\{ x \in \boldsymbol{R}^n : u(x) > \frac{1}{2} \right\}.$$

Threshold Dynamics

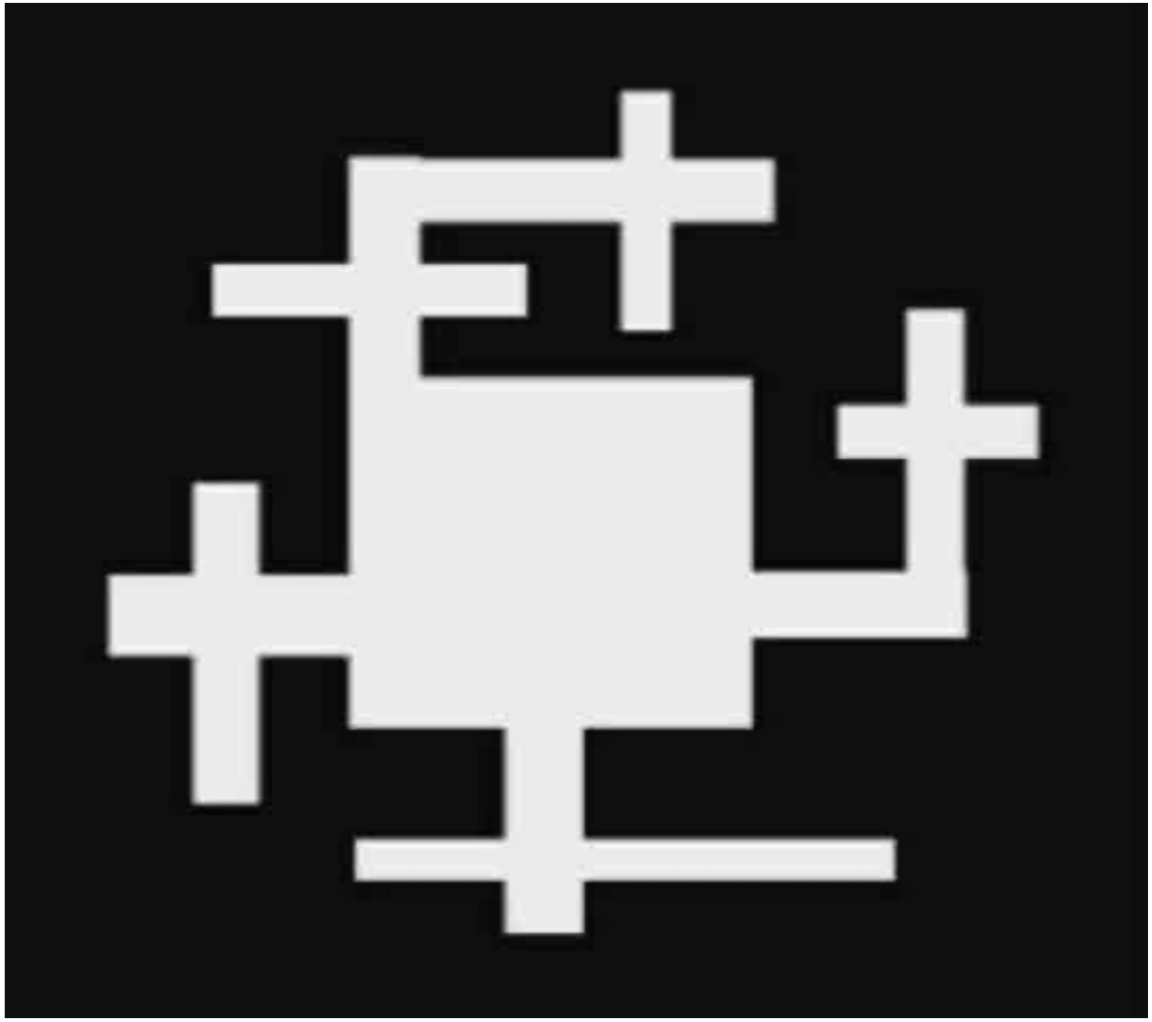
Benefits:

- Unconditionally stable:
  - Accuracy only concern in choosing time step size.
- Fast implementation:
  - Complexity only $O(N \log N)$, where N is the total number of grid points.
- Implicit representation of the front, allowing automatic topology changes.

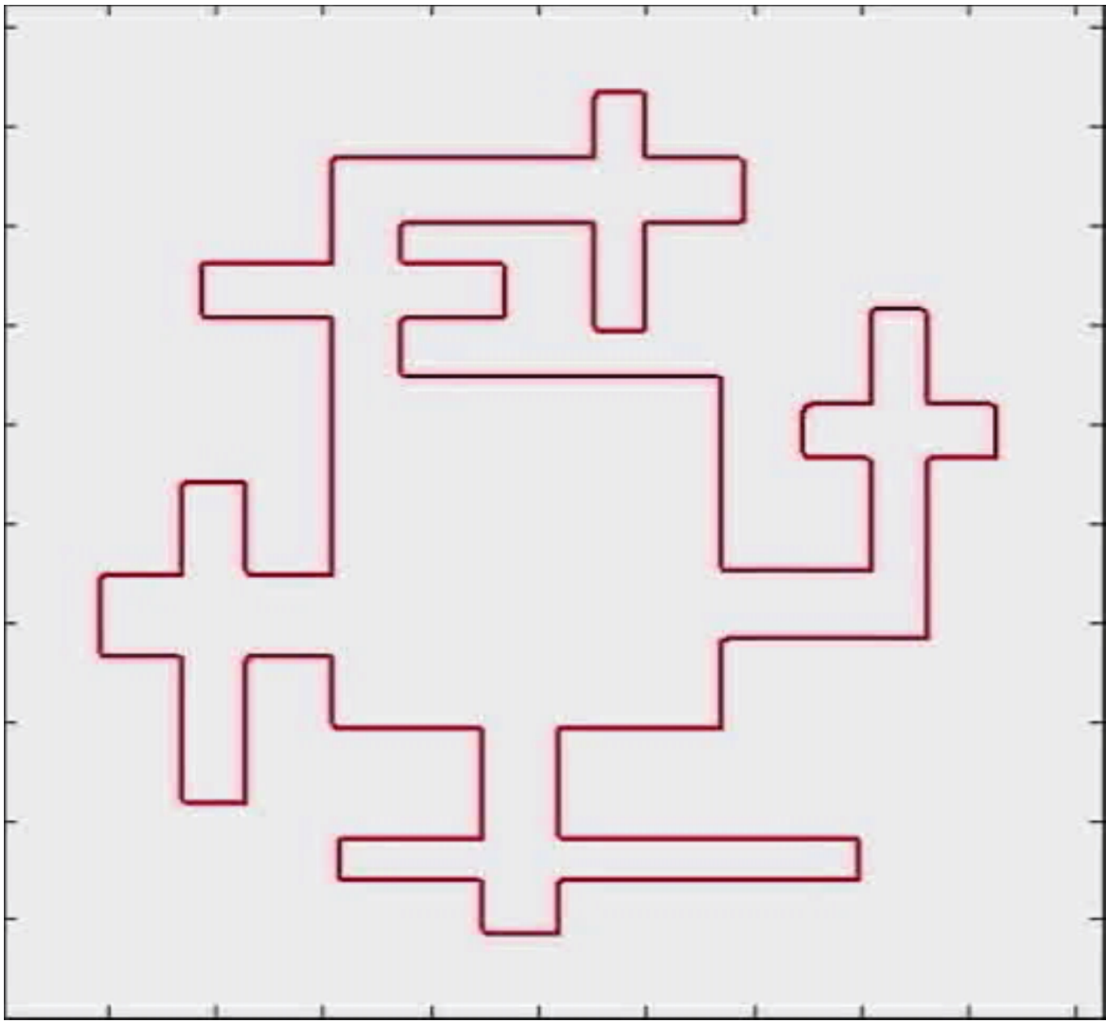
Caveats:

- No sub-grid accuracy.
- Dynamics can be very inaccurate on uniform grids.
- Can get "stuck" if δt too small.

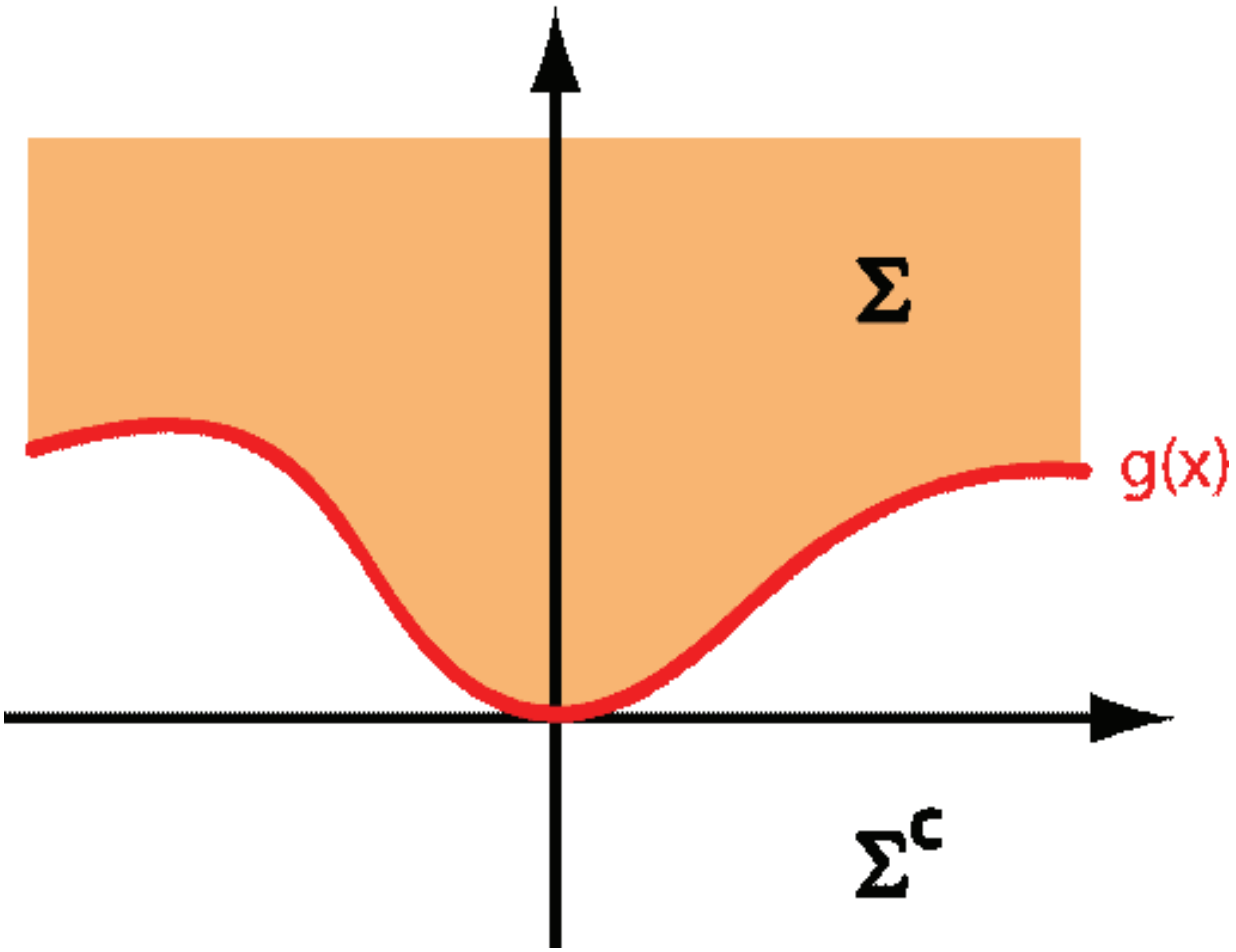
Threshold Dynamics



Threshold Dynamics



Threshold Dynamics



Threshold Dynamics

- Mascarenhas calculated the following: Let

$$F(x, y, t) = (\mathbf{1}_{\Sigma} * G_t)(x, y).$$

- Express the convolution in terms of $g(\xi)$.
- Taylor expand $F(0, y, t)$ at $y = 0$:

$$F(0, y, t) = \frac{1}{2} - \frac{1}{2\sqrt{\pi}} y t^{-\frac{1}{2}} + \frac{1}{2\sqrt{\pi}} g''(0) t^{\frac{1}{2}} + O\left(t^{\frac{3}{2}}\right)$$

  provided that $y = O(t)$ as $t \to 0^+$.

- At the origin, we have

$$g_{ss} = g'' = \kappa$$

Threshold Dynamics

- Thresholding is equivalent to setting

$$F(0, y, t) = \frac{1}{2}$$

which gives

$$\frac{1}{2} = F(0, y, t) \approx \frac{1}{2} - \frac{1}{2\sqrt{\pi}} y\, t^{-\frac{1}{2}} + \frac{1}{2\sqrt{\pi}} \kappa t^{\frac{1}{2}}$$

- Solving for y:

$$y \approx \kappa t.$$

- UPSHOT: Interface moves with normal speed κ for duration t in the normal direction.

Threshold Dynamics

- Mascarenhas also proposed the generalization:

$$v_n = \kappa + \alpha$$

1. Step 1: Convolution.

$$u_{k+1} = (\mathbf{1}_{\Sigma_k} * G_t)$$

2. Step 2: Dilation.

$$u_{k+1} \to u_{k+1} + \alpha\sqrt{\delta t}$$

3. Step 3: Thresholding.

$$\Sigma_{k+1} = \left\{x : u_{k+1}(x) \geq \frac{1}{2}\right\}.$$

Threshold Dynamics for Mumford-Shah

- Recall normal speed for P.C. Mumford-Shah:

$$v_n = \kappa + \lambda\big((f - c_2)^2 - (f - c_1)^2\big)$$

- One option for threshold dynamics:

1. Step 1: Convolution.

$$u_{k+1} = \mathbf{1}_{\Sigma_k} * G_{\delta t}$$

2. Step 2: Dilation.

$$u_{k+1} \to u_{k+1} + \lambda\sqrt{\delta t}\big((f - c_2)^2 - (f - c_1)^2\big)$$

3. Step 3: Thresholding.

$$\Sigma_{k+1} = \left\{x : u_{k+1} \geq \frac{1}{2}\right\}$$

- Doesn't work well: Time splitting error too large for large λ.

Threshold Dynamics for Mumford-Shah

- Joint work with Richard Tsai.
- Time split the phase field approximation of P.C. Mumford-Shah:

$$u_t = \varepsilon \Delta u - \frac{1}{\varepsilon} W'(u) - \lambda (f - c_1)^2 u + \lambda (f - c_2)^2 (1 - u)$$

1. Step 1: Solve a linear PDE:

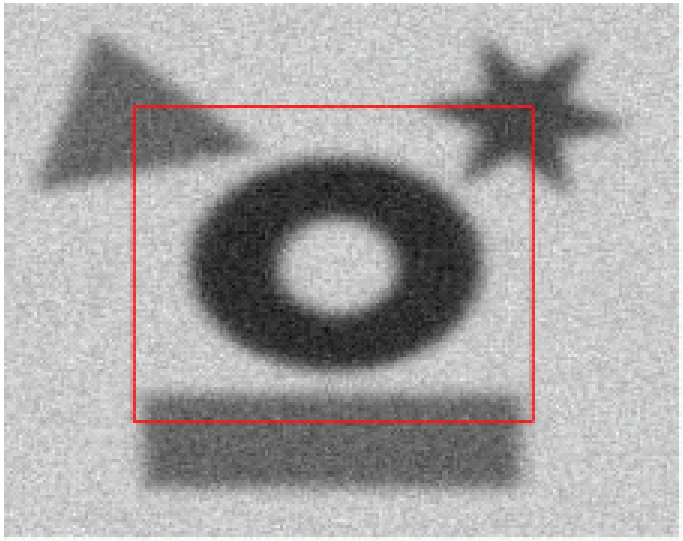
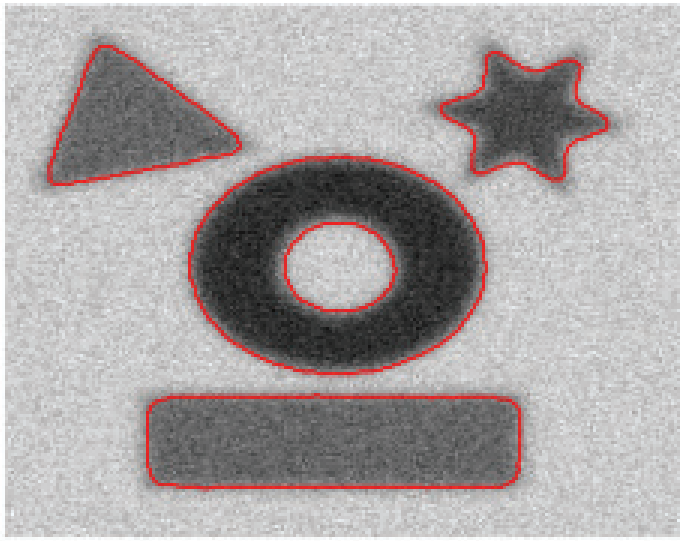
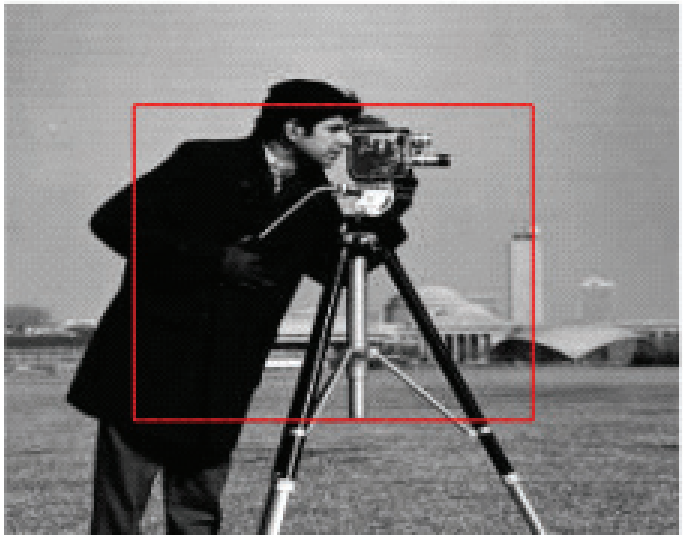
$$v_t = \Delta v - \frac{\lambda}{\sqrt{\pi \delta t}} \left(v (c_1 - f)^2 + (v - 1)(c_2 - f)^2 \right)$$

$$v(x, 0) = u_n(x)$$

2. Step2: Threshold:

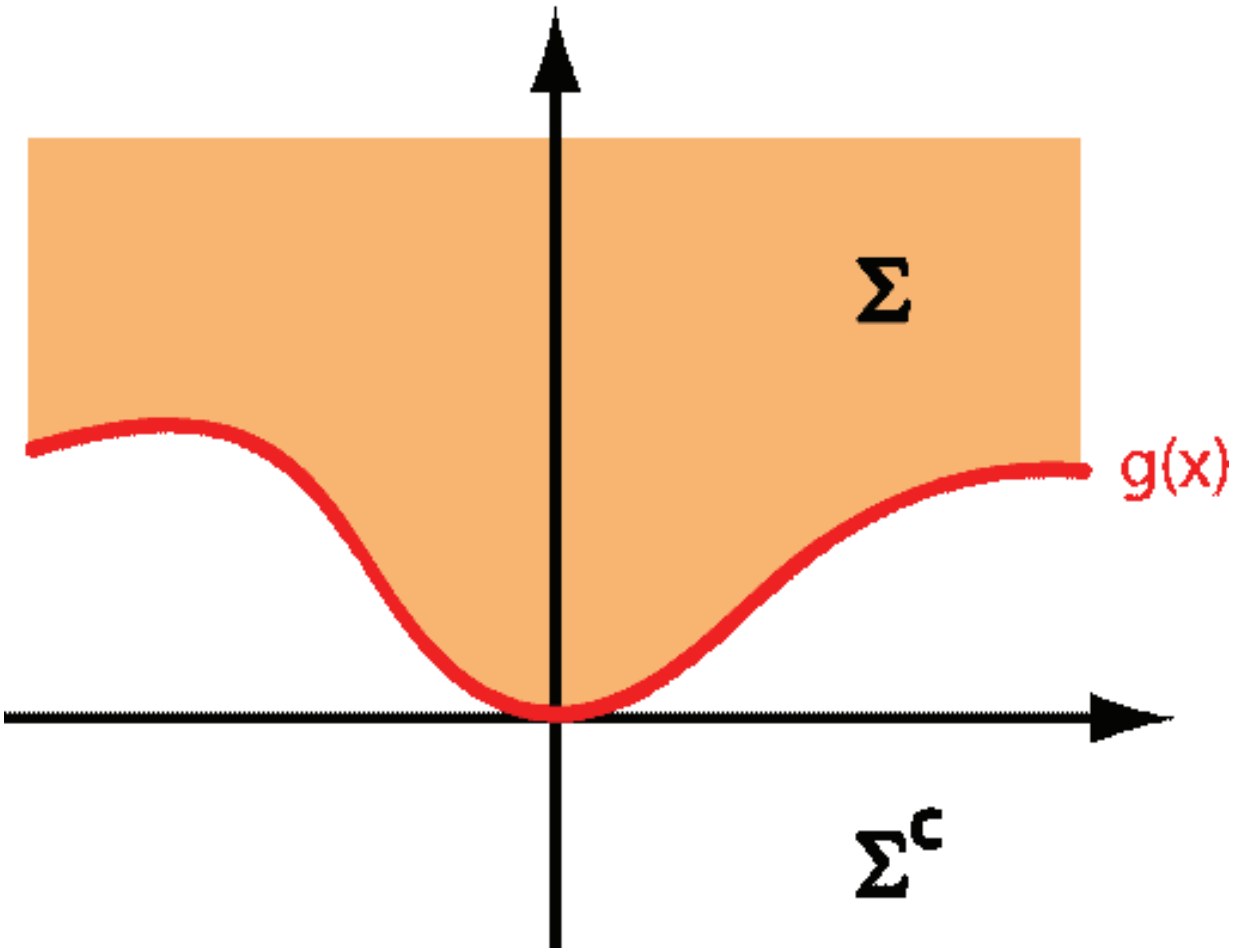
$$u_{n+1}(x) = \begin{cases} 0 & \text{if } \; v(x, \delta t) \leq \frac{1}{2}, \\ 1 & \text{if } \; v(x, \delta t) > \frac{1}{2}. \end{cases}$$

Threshold Dynamics for Mumford-Shah


These computations take just a few iterations.

Threshold Dynamics: Extensions



Threshold Dynamics: Extensions

- For the elastica energy

$$\int_{\partial\Sigma} \kappa^2 \, d\sigma$$

- L^2 gradient descent is known as Willmore flow.
- Normal speed of the curve:

$$v_n = \boxed{W = -\kappa_{ss} - \frac{1}{2}\kappa^3.}$$

- Extension of threshold dynamics to Willmore flow:
  - Grzibovskis, Heintz. *A convolution thresholding scheme for the Willmore functional*. Interfaces and Free Boundaries (2008).

Threshold Dynamics: Extensions

- Following Mascarenhas (1992), Ruuth (1996), let

$$F(x, y, t) = (\mathbf{1}_{\Sigma} * G_t)(x, y)$$

- Express the convolution in terms of $g(\xi)$.
- Taylor expand $g(\xi)$ at $\xi = 0$:

$$F(0, y, t) = \frac{1}{2} - \frac{1}{2\sqrt{t}} y t^{-\frac{1}{2}} + \frac{1}{2\sqrt{t}} g''(0) t^{\frac{1}{2}} + O\left(t^{\frac{3}{2}}\right)$$

provided that $y = O(t)$ as $t \to 0$.

Threshold Dynamics: Extensions

- If we continue Mascarenhas' expansion, we find:

$$F(0,y,t) = \frac{1}{2} - \frac{1}{2\sqrt{t}} y t^{-\frac{1}{2}} + \frac{1}{2\sqrt{t}} g''(0) t^{\frac{1}{2}}$$

$$+ \frac{1}{4\sqrt{\pi}} g^{(4)}(0) t^{\frac{3}{4}} - \frac{5}{8\sqrt{\pi}} (g''(0))^3 t^{\frac{3}{2}} + O(t^{\frac{5}{2}})$$

provided once again that $y = O(t)$.

- Error term in curvature motion to leading order is:

$$\left(\frac{1}{4\sqrt{\pi}} g^{(4)}(0) - \frac{5}{8\sqrt{\pi}} (g''(0))^3 \right) t.$$

Threshold Dynamics: Extensions

- At the origin,

$$g''(0) = g_{ss}(0) = \kappa(0),$$

and

$$g^{(4)}(0) = \kappa_{ss} + 3\kappa^3.$$

Therefore,

$$W = -g^{(4)}(0) + \frac{5}{2}\kappa^3(0).$$

That means:

$$F(0, y, t) = \frac{1}{2} - \frac{1}{2\sqrt{\pi}} y t^{-\frac{1}{2}} + \frac{1}{2\sqrt{\pi}} \kappa t^{\frac{1}{2}} + \frac{1}{2\sqrt{\pi}} W t^{\frac{3}{2}} + O\left(t^{\frac{5}{2}}\right).$$

Threshold Dynamics: Extensions

- Grzibovskis & Heintz on Willmore flow:
  - Take convolutions using two different Gaussians,
  - Take the correct linear combination between the two convolutions so that lower order curvature terms cancel out.
  - Threshold.
- Algorithm:

1. Form the convolution:

$$A(x,y) = (2\delta t)^{\frac{1}{4}}\mathbf{1}_{\Sigma^k} * \left(2G_{\frac{\sqrt{2\delta t}}{4}} - \frac{1}{2}G_{4\sqrt{2\delta t}}\right)$$

2. Set:

$$\Sigma^{k+1} = \left\{(x,y) : A(x,y) \geq \frac{3(2\delta t)^{\frac{1}{4}}}{4}\right\}$$

Threshold Dynamics: Extensions

- Joint work with Ruuth and Tsai:
  - Esedoglu, Ruuth, Tsai. *Threshold dynamics for disocclusion and shape reconstruction.* Proc. of ICIP (2005).
  - Esedoglu, Ruuth, Tsai. *Threshold dynamics for high order geometric motions.* Interfaces and Free Boundaries (2008).
- Extensions to:
  - Willmore + Lower order terms:

$$v_n = c_0(x, y, t) + c_{1(x,y,t)}\kappa + W$$

  - Surface diffusion:

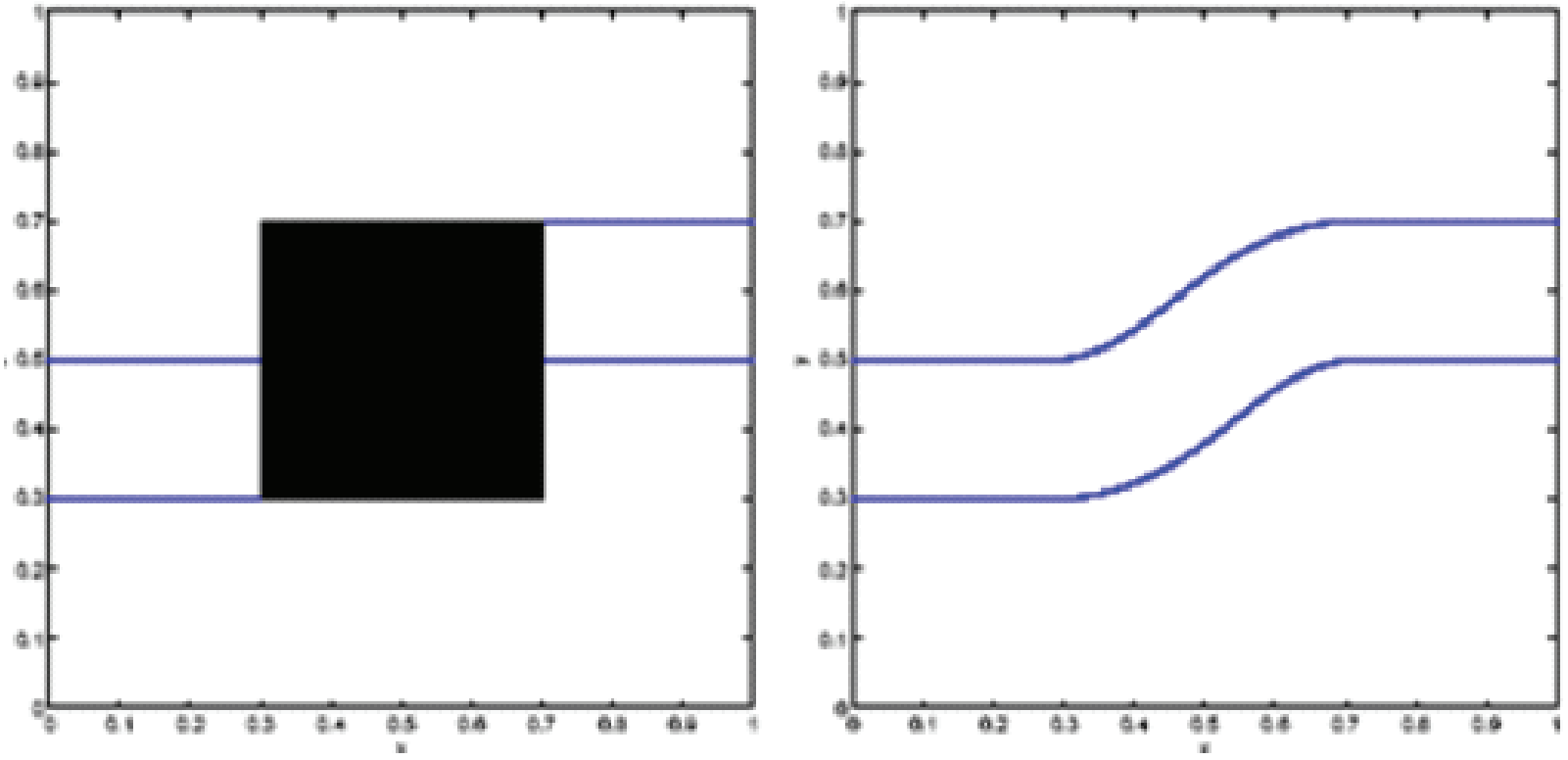
$$v_n = \kappa_{ss}.$$

- Applications to inpainting.

Threshold Dynamics: Extensions

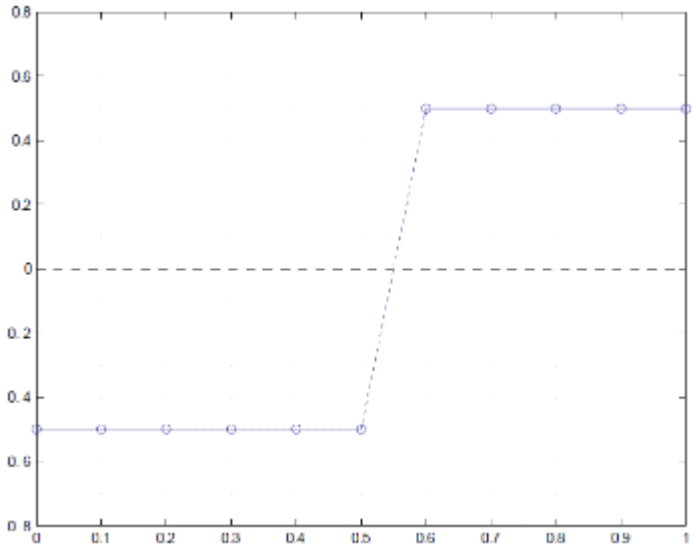
An inpainting example that involves the normal speed:

$$v_n = W + \alpha\kappa + \beta(x, y).$$



Distance Function Dynamics

- Joint work with Ruuth and Tsai (2009):
  - Esedoglu, Ruuth, Tsai. *Diffusion generated motion using the signed distance function.* Journal of Computational Physics (2009).
- Motivation:
  - Threshold dynamics is very inaccurate on uniform grids.
  - Cannot interpolate to locate interface at subgrid resolution.



Distance Function Dynamics

- New version of the algorithm:
  - Represent Σ not by $\mathbf{1}_\Sigma(x)$, but by its signed distance function.

$$d_\Sigma(x) = \begin{cases} \inf_{y \in \Sigma^c} |x - y| & \text{if } x \in \Sigma, \\ -\inf_{y \in \Sigma} |x - y| & \text{if } x \in \Sigma^c. \end{cases}$$

- Alternate two steps:

1. Convolution:

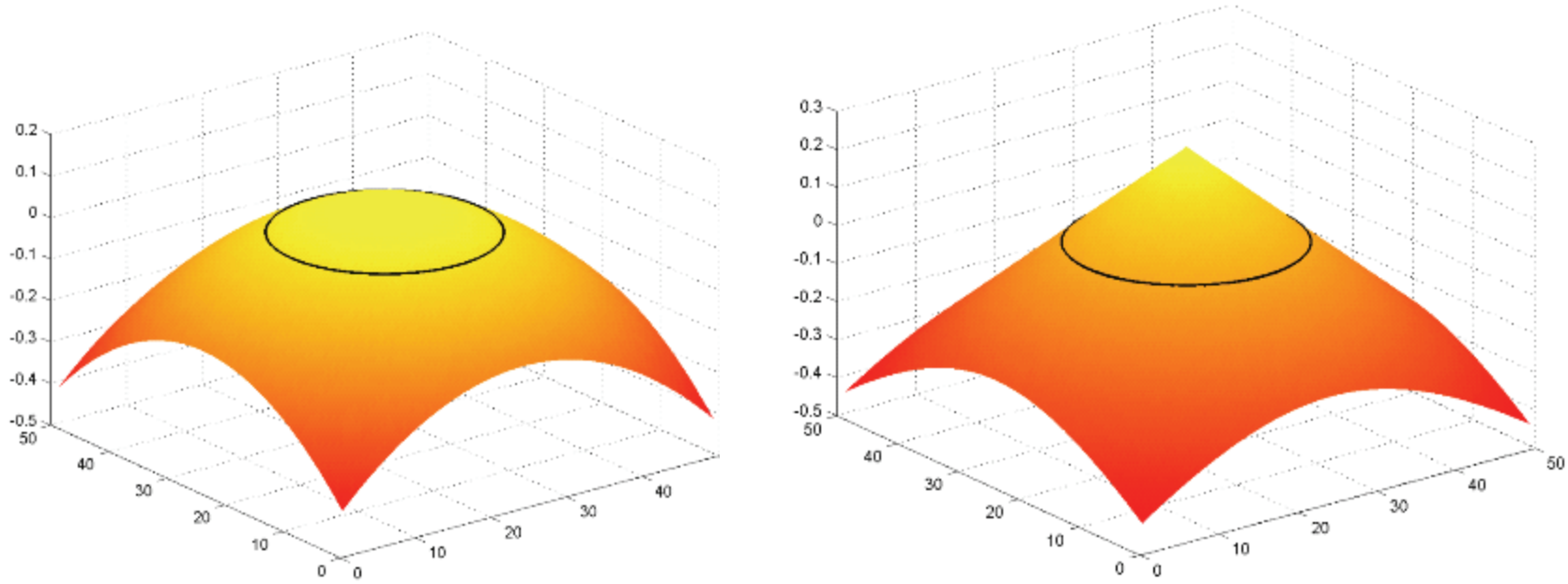
$$u(x) = G_{\delta t} * d_\Sigma$$

2. Redistancing:

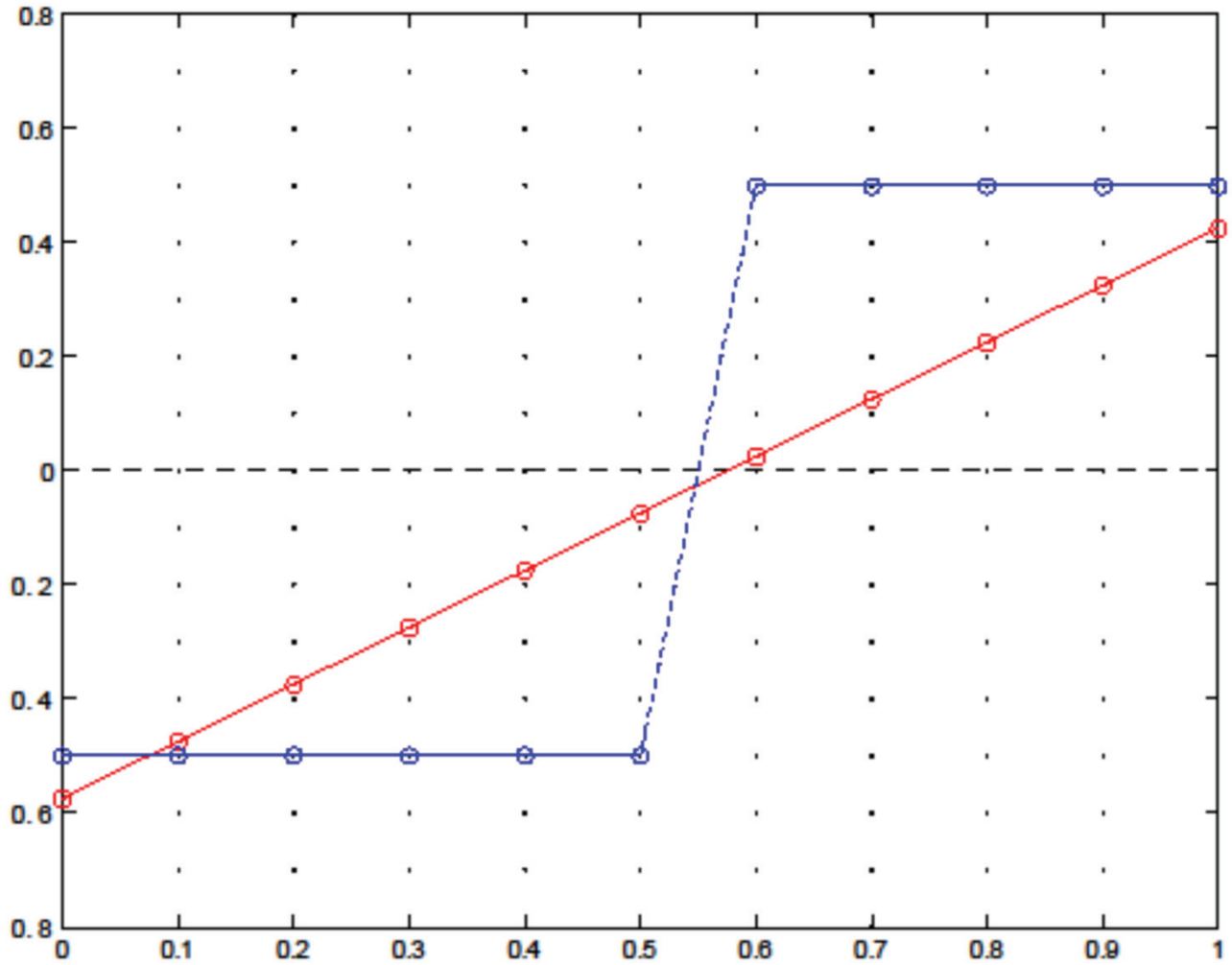
$$d_{\Sigma^{k+1}}(x) = \mathbf{Redist}(u) := \begin{cases} \inf_{\{y : u(y) > 0\}} |x - y| & \text{if } u(x) > 0, \\ -\inf_{\{y : u(y) < 0\}} |x - y| & \text{if } u(x) < 0. \end{cases}$$

Distance Function Dynamics

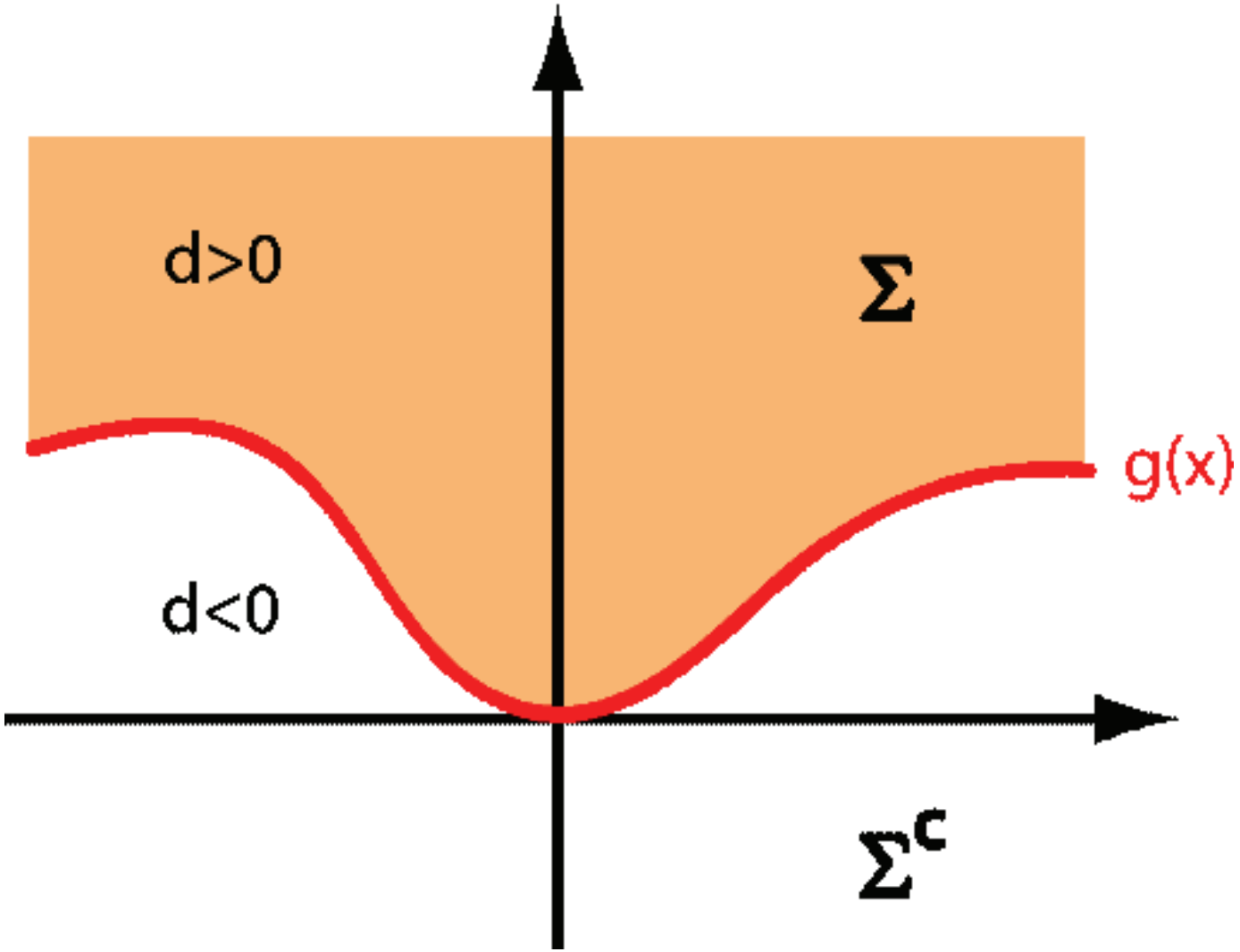
- Standard tool in level-set methods.
  1. Keeps 0-level set $\partial\Sigma = \{x : \phi(x) = 0\}$ of $\phi(x)$ fixed,
  2. Evolves $\phi(x,t)$ towards $d_\Sigma(x)$.
- $O(N \log N)$ algorithms exist: Fast marching, fast sweeping, etc.



Distance Function Dynamics



Distance Function Dynamics



Distance Function Dynamics

We must expand $d(x,y)$ at $(x,y)=(0,0)$. We know the following:

- Eikonal equation:

$$\boxed{d_x^2(x,y)+d_y^2(x,y)=1}$$

  in a neighborhood of $\partial\Sigma$.

- Boundary condition:

$$\boxed{d(x,g(x))=0}$$

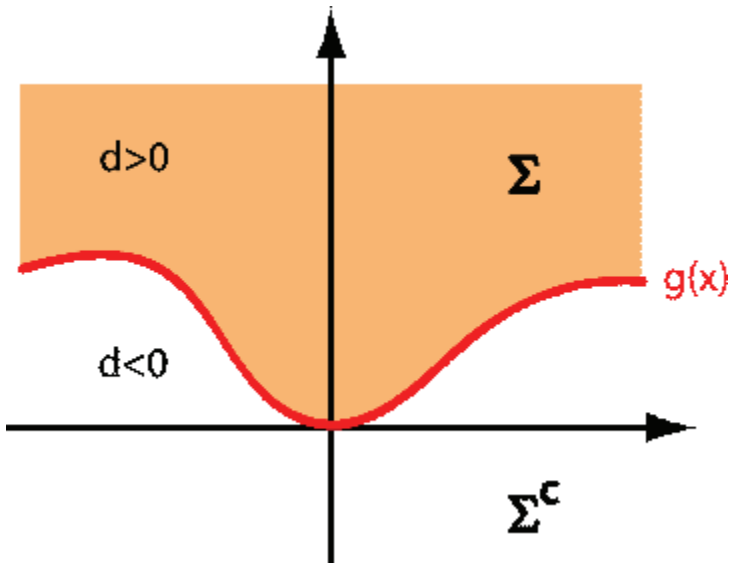
- Curvature and $d(x,y)$:

$$\boxed{d_{xx}(x,g(x))+d_{yy}(x,g(x))=\kappa(x)}$$

Distance Function Dynamics

Moreover, for sufficiently small y:

$$d(0,y) = y, \text{ so that}$$
$$d_y(0,y) = 1,$$
$$d_{yy}(0,y) = 0,$$
$$d_{yyy}(0,y) = 0,$$
$$\text{etc...}$$



Distance Function Dynamics

Let

$$A(x,y) = d_x^2(x,y) + d_y^2(x,y).$$

Then,

$$A_x = 2d_x d_{xx} + 2d_y d_{xy} = 0,$$
$$A_y = 2d_x d_{xy} + 2d_y d_{yy} = 0.$$

And,

$$A_{xx} = 2d_{xx}^2 + 2d_x d_{xx} + 2d_{xy}^2 + 2d_y d_{xxy} = 0.$$

Also, use:

$$d_{xx}(x, g(x)) + d_{yy}(x, g(x)) = \kappa(x).$$

Distance Function Dynamics

Differentiating these expressions many times, taking linear combinations, we arrive at

$$d_x(0, y) = 0,$$
$$d_{xy}(0, y) = 0,$$
$$d_{xyy}(0, y) = 0$$

for all sufficiently small y.

Distance Function Dynamics

Keep differentiating. In the same fashion, we get

$$
\begin{aligned}
d_{xxy}(0,0) &= -\kappa^2(0), \\
d_{xxx}(0,0) &= \kappa_x(0), \\
d_{xxxy}(0,0) &= -3\kappa(0)\kappa_x(0), \\
d_{xxyy}(0,0) &= 2\kappa^3(0), \\
d_{xxxx}(0,0) &= \kappa_{xx}(0) - 3\kappa^3(0), \\
&\text{etc...}
\end{aligned}
$$

Distance Function Dynamics

These lead to the following expansion of $d(x,y)$ at $(0,0)$:

Expansion of $d(x,y)$

$$\begin{aligned} d(x,y) = y & \\ & + \frac{1}{2}\kappa x^2 \\ & + \frac{1}{6}(\kappa_x x^3 - 3\kappa^2 x^2 y) \\ & + \frac{1}{24}\left((\kappa_{xx} - 3\kappa^3)x^4 - 12\kappa\kappa_x x^3 y + 12\kappa^3 x^2 y^2\right) \\ & + o\left((x^2+y^2)^2\right). \end{aligned}$$

Distance Function Dynamics

Expansion of $d(x,y)$ leads to an expansion for its convolution with the kernel $G_t(x)$:

Expansion of the Convolution

$$\begin{aligned}(G_t * d)(0,y) &= y \\ &\quad + \kappa t \\ &\quad - \kappa^2 ty + \frac{1}{2}\left(\kappa_{xx} - \kappa^3\right) t^2 \\ &\quad + o(t^2).\end{aligned}$$

Distance Function Dynamics

$$0 = (G_t * d)(0, y) \approx y + \kappa t \;\Rightarrow\; y \approx -\kappa t.$$

Algorithm

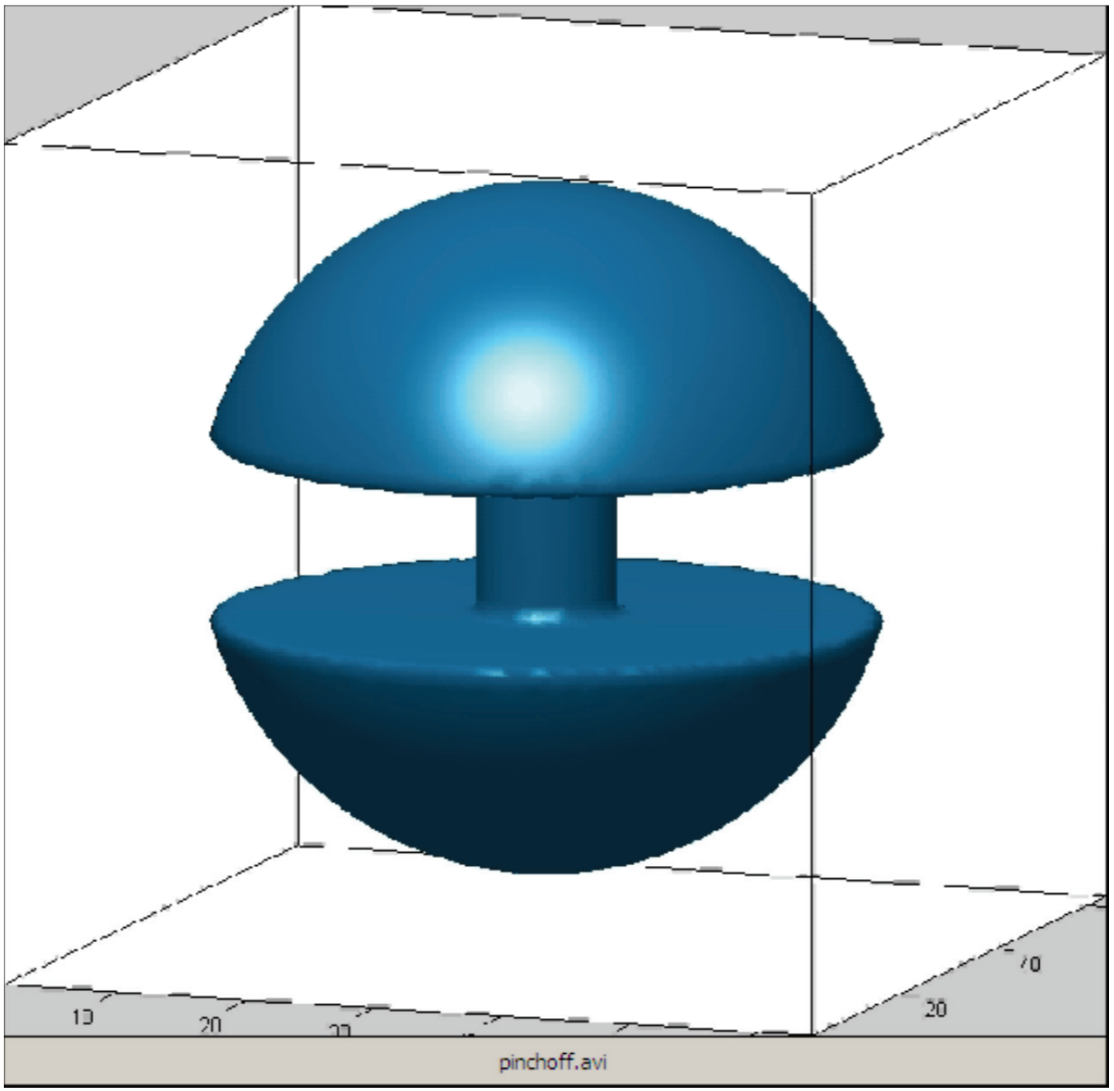
1. Convolution step: For $k = 0, 1, 2, \ldots$ let

$$L^k(x) = \left(G_{\delta t} * d^k\right)(x)$$

2. Redistancing step:

$$d^{k+1}(x) = \textbf{Redist}(L^k)$$

Distance Function Dynamics



Distance Function Dynamics

Notes:

- Expansions suggest first order accuracy in time.
- The scheme is unconditionally monotone:
  - Convolution with positive kernel preserves order,
  - Signed distance functions preserve order.
- Convergence to viscosity solution (in non-fattening situations) proven by Chambolle and Novaga.

Distance Function Dynamics

High order in time, multi-step algorithm for curvature motion:

Algoritm

1. Convolution step: For $k = 1, 2, \ldots$ form the functions

$$A_1(x) = G_{2\delta t} * d^{k-1},$$
$$A_2(x) = G_{\delta t} * d^{k}.$$

2. Redistancing step:

$$d^{k+1}(x) = \textbf{Redist}\left(\frac{1}{3}(4A_2 - A_1)\right).$$

Note:

- Second order accurate in time.
- One redistancing per time step.

Distance Function Dynamics

Convergence study with multistep algorithm:

Resolution	# of Time Steps	Relative Error	Order
32×32	10	0.092%	–
64×64	20	0.027%	1.77
128×128	40	0.0078%	1.79
256×256	80	0.0020%	1.96
512×512	160	0.00052%	1.94

Error in the radius of a shrinking circle.

Distance Function Dynamics

An algorithm for $v_n = f(\kappa)$

Algorithm

1. Convolution step: For $k = 0, 1, 2, \ldots$ let

$$L^k(x) = d^k(x) + f\left(\frac{(G_{N\delta t} * d^k)(x) - d^k(x)}{N\delta t}\right)\delta t$$

2. Redistancing step:

$$d^{k+1}(x) = \mathbf{Redist}(L^k)$$

Notes:

- This is an unconditionally monotone algorithm.

Distance Function Dynamics

- Recall from our expansions:

$$(G_t * d)(0, y) = y + \kappa t + O(t^2)$$

provided that $y = O(t)$, and

$$d(0, y) = y$$

for all small enough y.

Distance Function Dynamics

- Isolate K:

$$G_{Nt} * d - y = G_{Nt} * d - d \approx N\kappa t$$

so that

$$\kappa \approx \frac{G_{Nt} * d - d}{Nt}.$$

- Now we have

$$f(\kappa) \approx f\left(\frac{G_{Nt} * d - d}{Nt}\right)$$

Distance Function Dynamics

- Then:

$$y + tf(\kappa) \approx d + f\left(\frac{G_{Nt} * d - d}{Nt}\right) t.$$

- Let

$$A(d) = d + f\left(\frac{G_{Nt} * d - d}{Nt}\right) t.$$

- Claim: If $d_1 \geq d_2$, and $N \geq \mathrm{Lip}_f$, then

$$A(d_1) \geq A(d_2).$$

Distance Function Dynamics

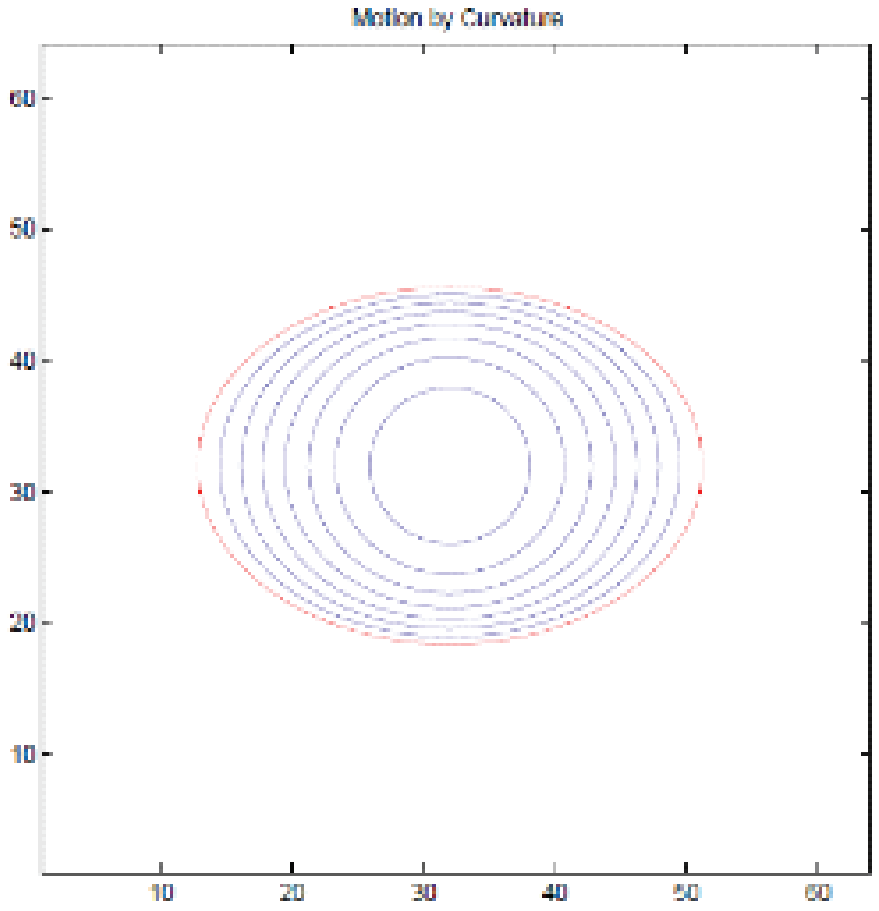
Example: Affine Invariant Curvature Motion

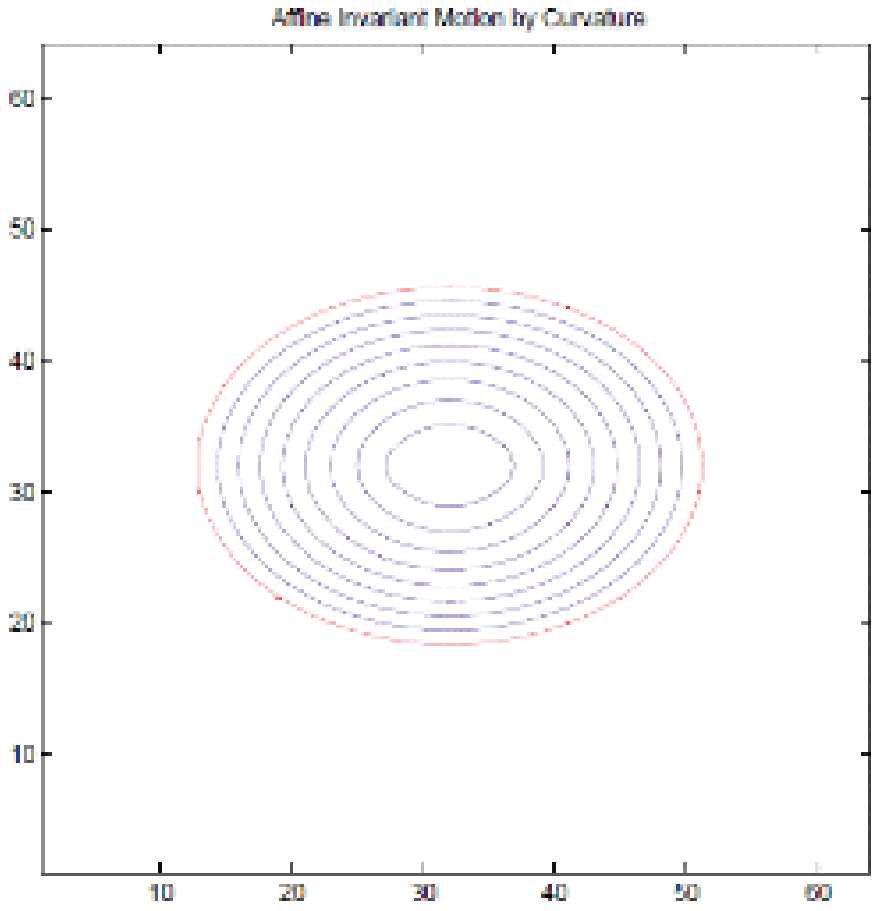
- Important in computer vision. Sapiro & Tannenbaum, others:

$$v_n = \kappa^{\frac{1}{3}}.$$

- Evolution commutes with affine transformations.
- In particular, an ellipse stays an ellipse with fixed eccentricity.
- Unfortunately, singular at $\kappa = 0$, but let's ignore that.

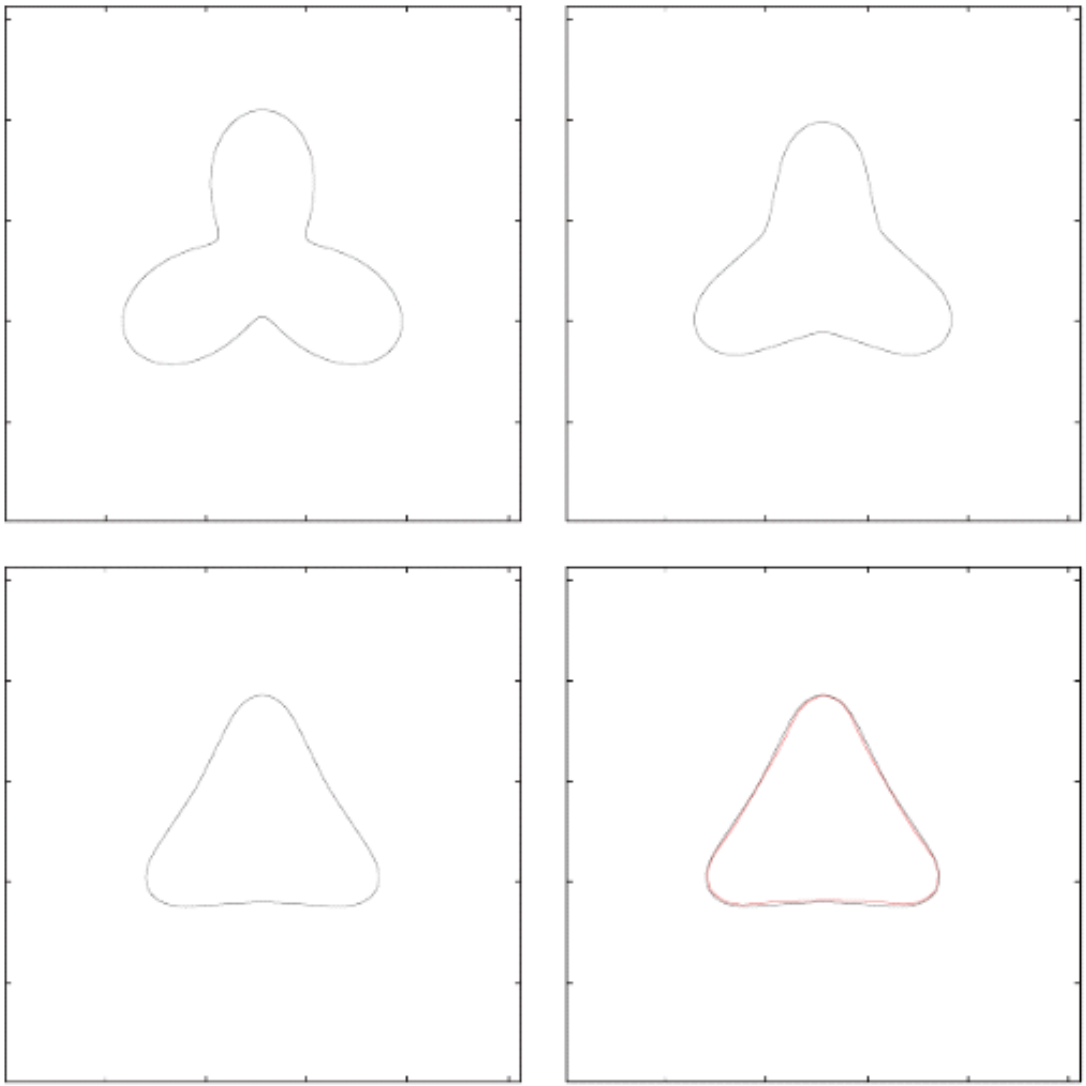
Distance Function Dynamics



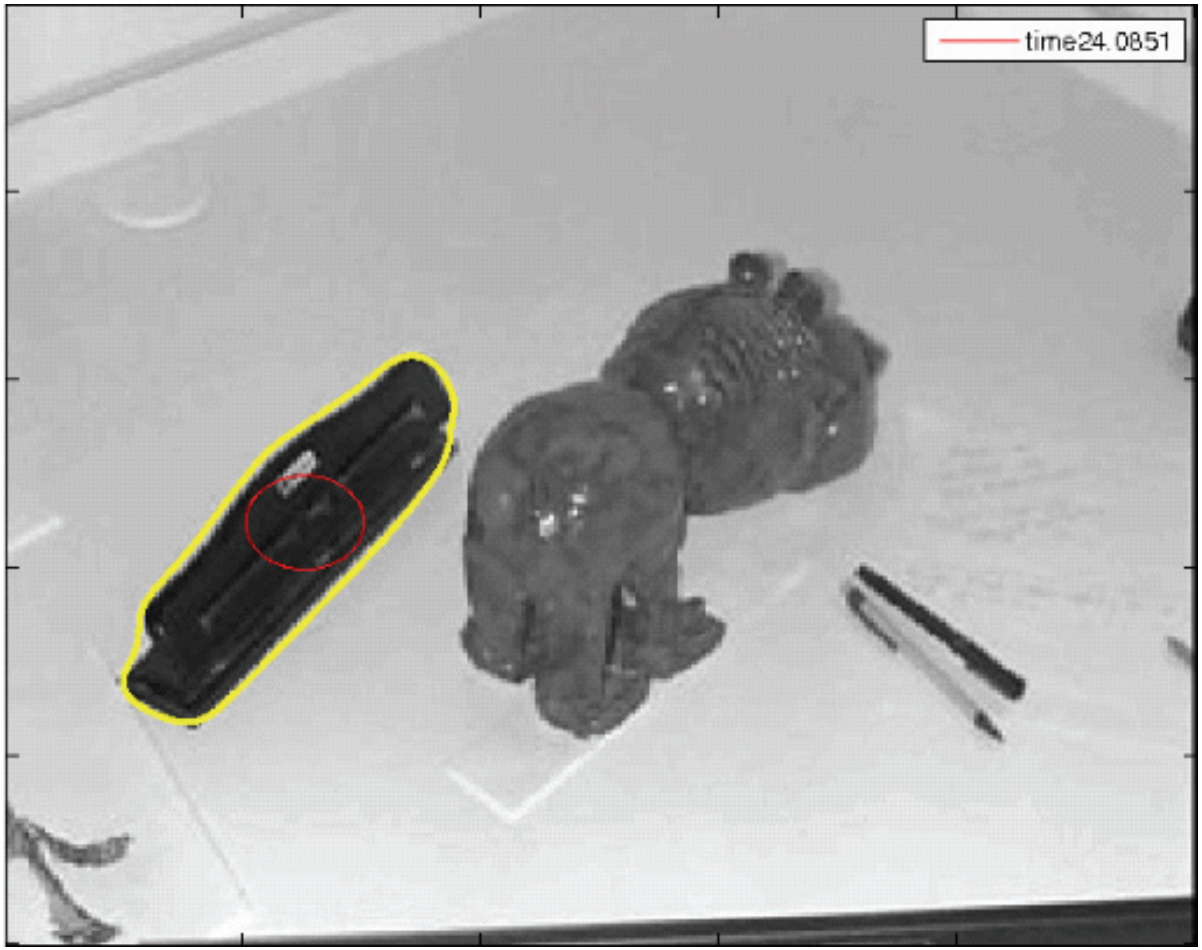


Distance Function Dynamics

Affine invariant curvature motion:

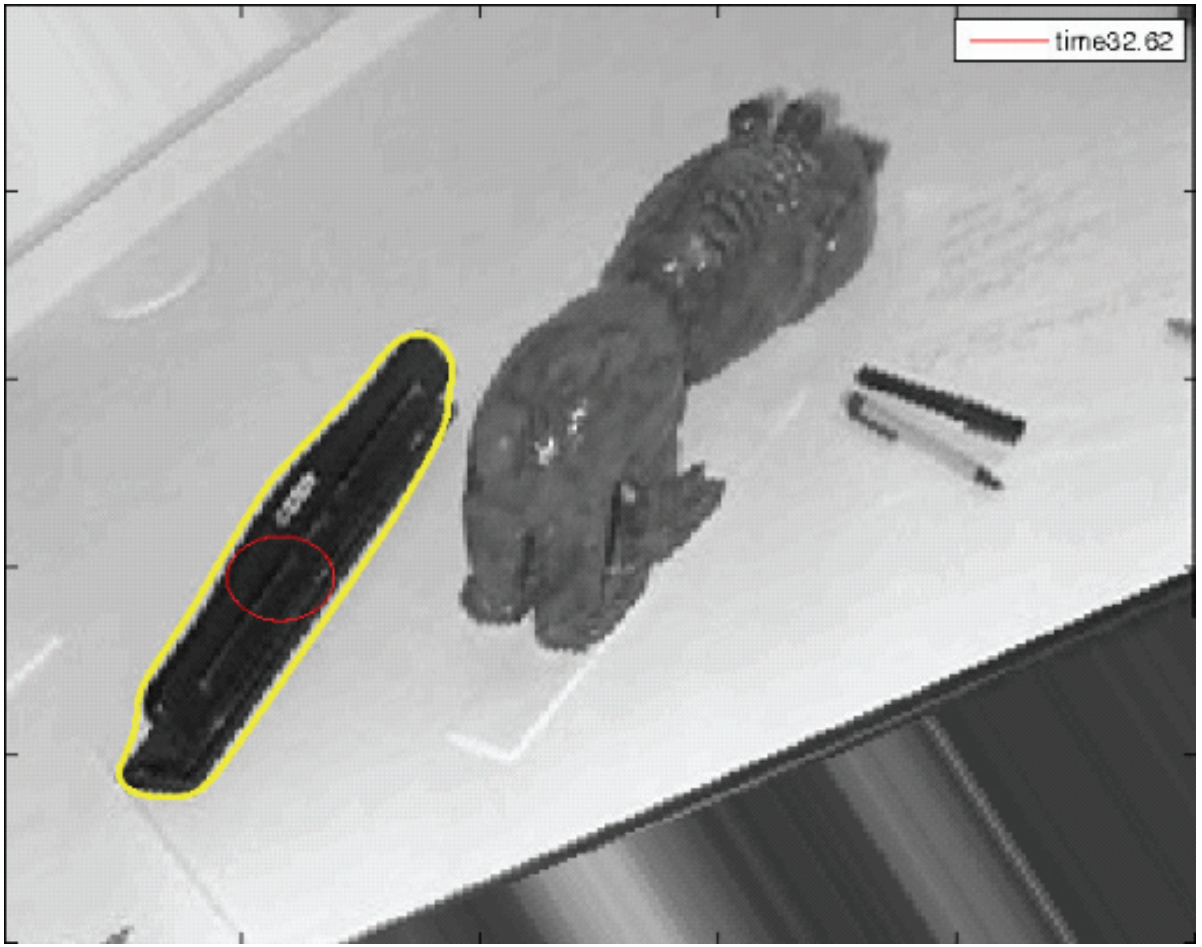


Distance Function Dynamics



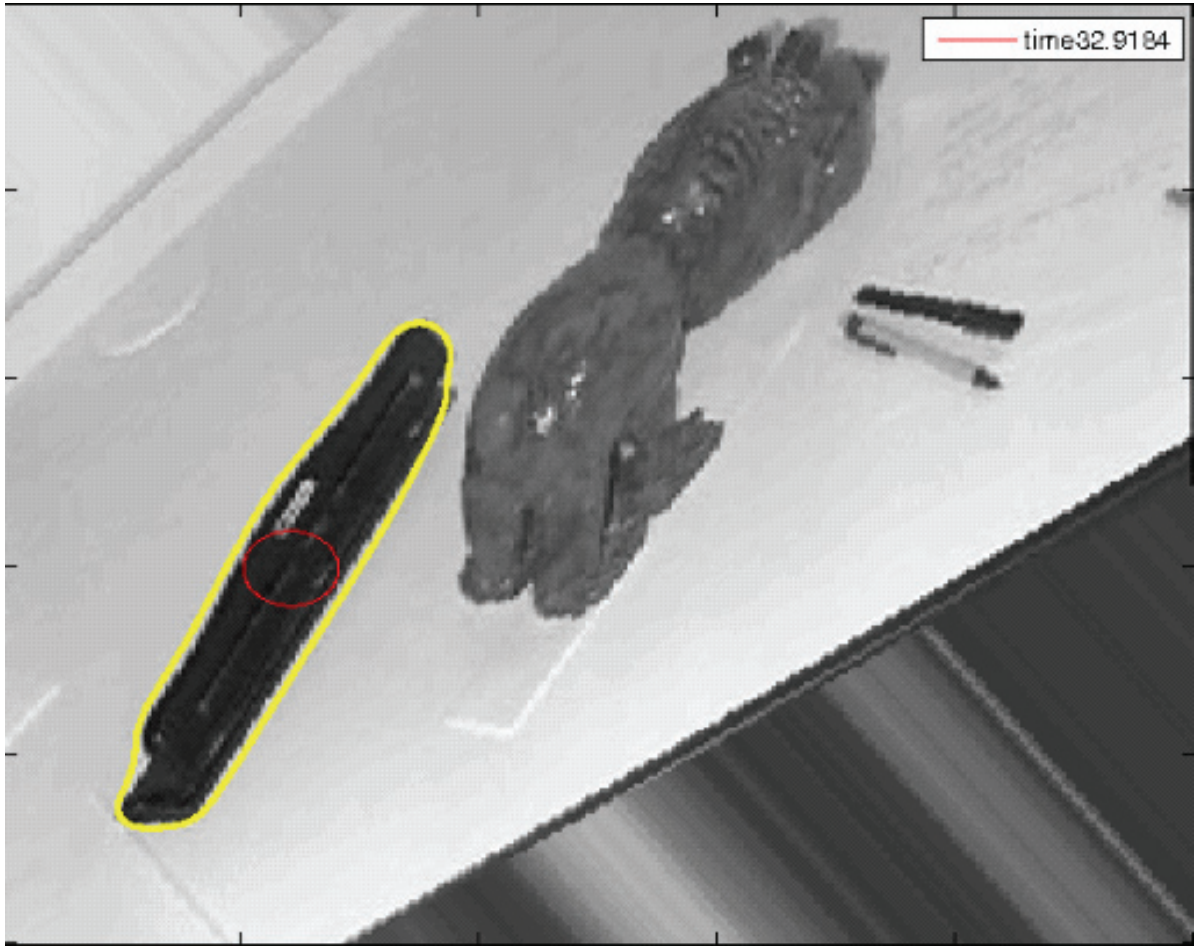
Work of Catherine Kublik

Distance Function Dynamics



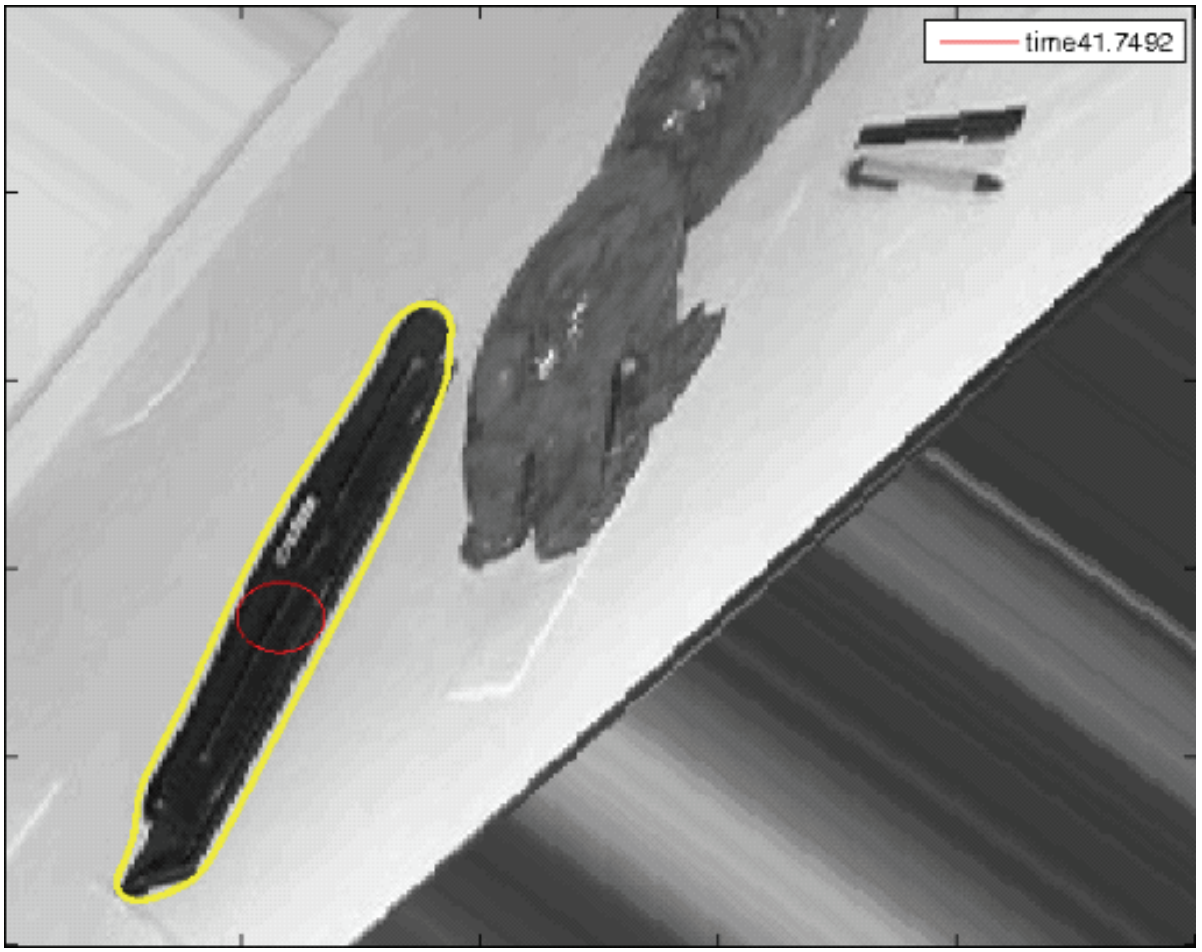
Work of Catherine Kublik

Distance Function Dynamics



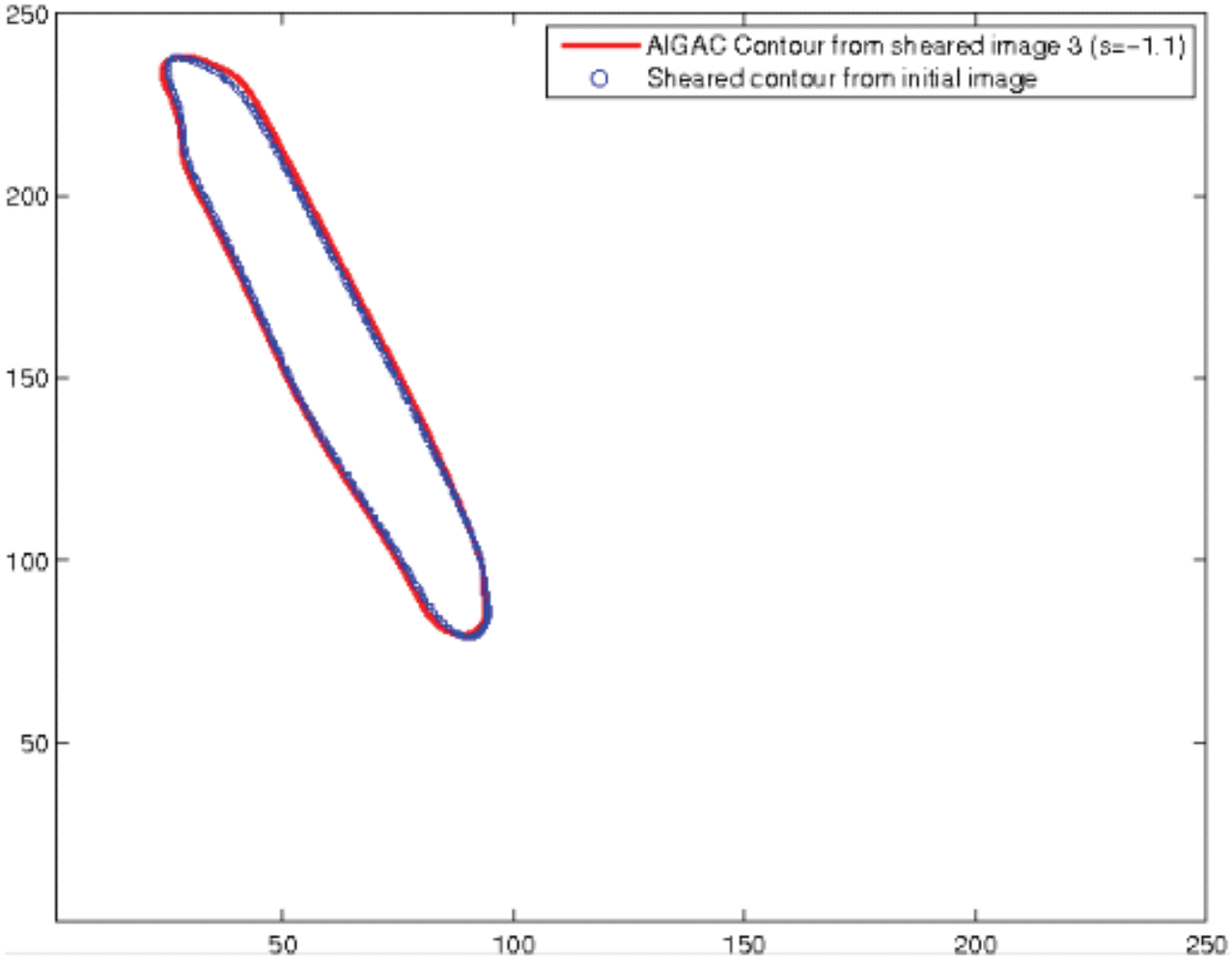
Work of Catherine Kublik

Distance Function Dynamics



Work of Catherine Kublik

Distance Function Dynamics



Work of Catherine Kublik

Distance Function Dynamics

High order motions via distance functions:

To generate 4th order flows, we cancel out the $O(t)$ terms in the expansion:

$$\Rightarrow \text{Take linear combinations of } G_{\sigma_1} * d \text{ and } G_{\sigma_2} * d$$

Example: Willmore flow. Recall:

$$\begin{aligned} W &= g_{xxxx} - \frac{5}{2}K^3 \\ &= K_{xx} - \frac{5}{2}K^3 \end{aligned}$$

Distance Function Dynamics

- Based on our expansions, we get:

$$\left(2G_{\sqrt{t}} - G_{2\sqrt{t}}\right) * d = y - t\left(K_{xx} - K^3\right) + o(t)$$

  at $x = 0$.

- We need to compensate by a factor of K^3:

$$\frac{1}{\sqrt{t}}\left(G_{\sqrt{t}} * d - d\right)^3 = K^3 t + o(t).$$

Distance Function Dynamics

We get an algorithm for Willmore flow:

Algorithm

1. Construct the signed distance function $d_0(x)$ to $\partial\Sigma_0$.
2. For $n = 0, 1, 2, \ldots$ let

$$L_n(x) = \left(G_{2\sqrt{t}} - 2G_{\sqrt{t}}\right) * d_n + \frac{C}{\sqrt{t}}\left(G_{\sqrt{t}} * d_n - d_n\right)^3$$

3. Construct the signed distance function $d_{n+1}(x)$ to the 0-level set of the function $L_n(x)$; go back to Step 2.

- Works in principle.
- Unfortunately, does not work very well in practice.

Dijkstra's Algorithm

Dijkstra's algorithm is formulated on graphs. We introduce some notation:

- Let $\{v_j\}$ denote the vertices of the graph.
- For each j, let S_j denote the star of the vertex v_j. These are all vertices connected to v_j by edges of the graph.
- For each j and $i \in S_j$, let $e_{j,i}$ denote the edge of the graph connecting v_j to v_i.
- For each j and $i \in S_j$, we will denote with $w_{j,i} \in \mathbb{R}$ the weight of the edge $e_{j,i}$.

Dijkstra's Algorithm

- We say that Γ is a path on the graph if it is an ordered list of vertices $\Gamma = (v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ such that

$$i_{j+1} \in S_{i_j} \text{ for each } j = 1, 3, \ldots, n-1.$$

- We define the length of the path Γ to be

$$L(\Gamma) = \sum_{j=1}^{n-1} w_{j\,j+1}.$$

- Let's write $\mathscr{P}(v_i, v_j)$ to denote all the paths on the graph that start at vertex v_i and end at vertex v_j.

Dijkstra's Algorithm

- Given two vertices v_i and v_j on the graph, we denote by $d(v_i, v_j)$ the distance between v_i and v_j, defined as follows:

$$d(v_i, v_j) = \min_{\Gamma \in \mathscr{P}(v_i, v_j)} L(\Gamma).$$

- We will also denote the distance function from the node v_i as

$$d_{v_i}(v_j) = d(v_i, v_j)$$

so that, in particular, $d_{v_i}(v_i) = 0$.

Dijkstra's Algorithm

- Known nodes are at which the distance function has been computed for sure.
- At the beginning of the algorithm, only the initial node v_i belongs to known nodes:

$$\mathsf{K} = \{i\} \text{ initially.}$$

- Trial nodes are the immediate neighbors of the known nodes:

$$\mathsf{T} = \bigcup_{j \in \mathsf{K}} S_j$$

so that

$$\mathsf{T} = S_i \text{ initially.}$$

- Far nodes are all the remaining nodes. They are assigned an temporary value of ∞ as their distances.

Dijkstra's Algorithm

- Dijkstra's algorithm progresses by switching one of the trial nodes from $\mathbf{T}$ to the known nodes class $\mathbf{K}$:
  - Step 1: Among the trial nodes in $\mathbf{T}$, find the one with the least temporary distance; call it v_{i_*}.
  - Step 2: Remove that node v_{i_*} from trial nodes group $\mathbf{T}$ and add it to known nodes group $\mathbf{K}$. Its temporary distance value becomes its final distance value:

$$\mathbf{K} \rightarrow \mathbf{K} \cup \{i_*\}, \mathbf{T} \rightarrow \mathbf{T} \setminus \{i_*\}, \text{ and } d_{v_i}(v_{i_*}) = \tilde{d}_{v_i}(v_{i_*}).$$

  - Step 3: Update the trial group $\mathbf{T}$: Since there is now a new known node (namely, v_{i_*}), there may be new trial nodes; add them to $\mathbf{T}$:

$$\mathbf{T} \rightarrow \mathbf{T} \cup (S_{i_*} \setminus \mathbf{K}) \text{ and } \mathbf{F} \rightarrow \mathbf{F} \setminus S_{i_*}.$$

Dijkstra's Algorithm

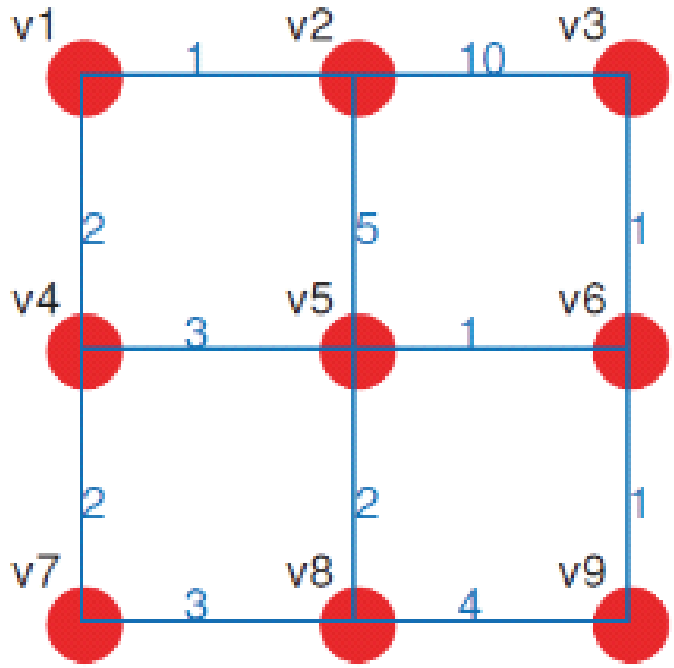
- Step 4: Update the temporary values of the trial neighbors of v_{i_*}:

  For each $j \in S_{i_*} \cap \mathbf{T}$, recompute $\tilde{d}_{v_i}(v_j) = \min_{k \in S_{v_j} \cap \mathbf{K}} \left(d_{v_i}(v_k) + w_{k,j} \right)$.

- The algorithm now repeats these steps, moving a single trial node from $\mathbf{T}$ to the set of known nodes $\mathbf{K}$ at every step; it therefore terminates in N steps, where N is the total number of nodes.

Dijkstra's Algorithm

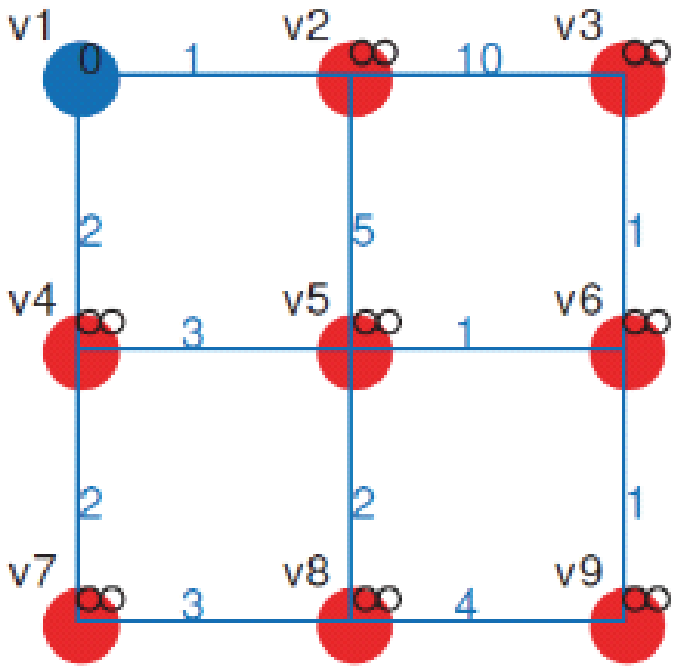
Example: We'll compute the distance function to node v_1 on the following graph:



RED = Far; GREEN = Trial; BLUE = Known.

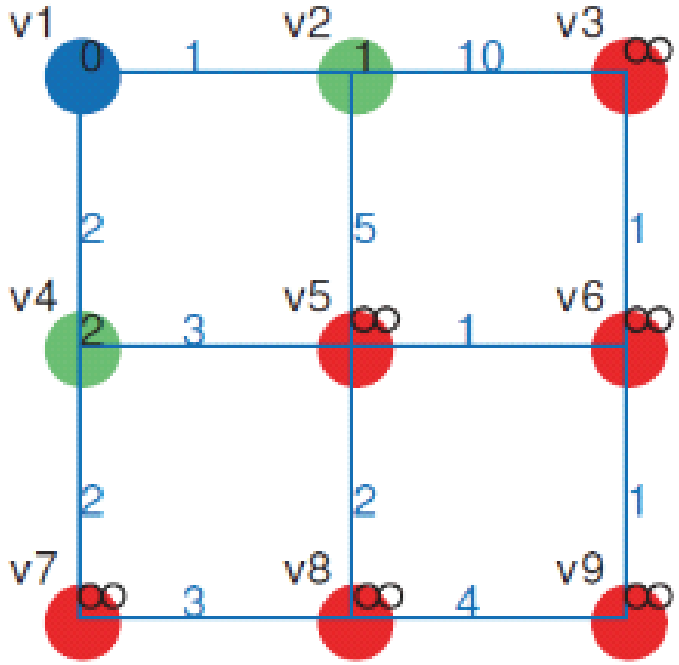
Dijkstra's Algorithm

In the beginning, only v_1 is known, whose distance is 0. All others are in the FAR group, with temp. values of ∞.



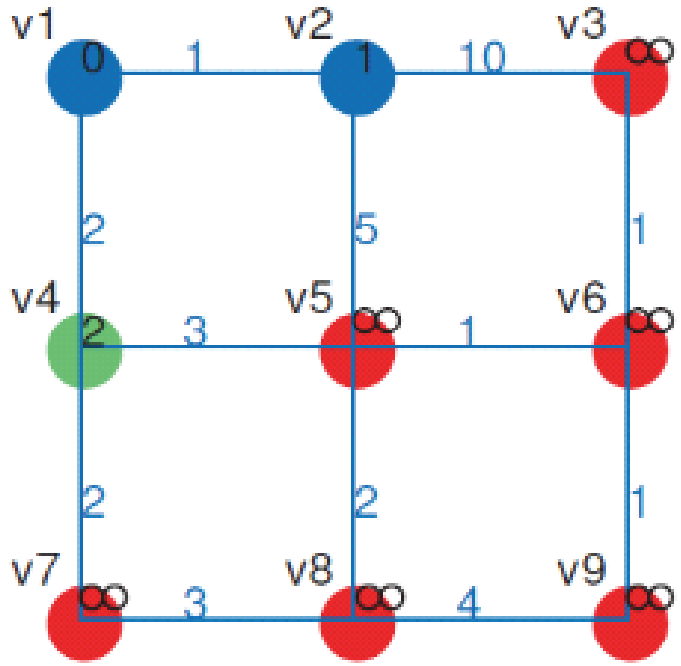
Dijkstra's Algorithm

The neighbors of v_1 are moved to the TRIAL group, and assigned temporary distance values.



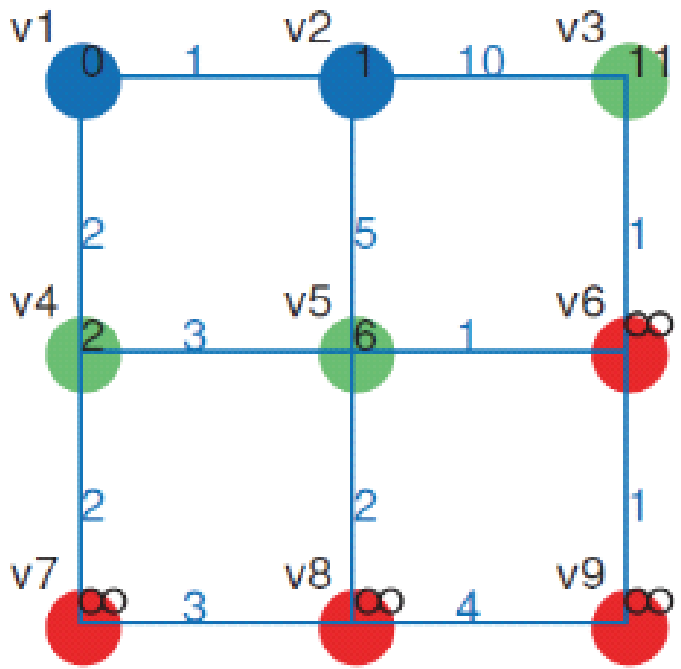
Dijkstra's Algorithm

v_2 is the smallest among the TRIAL nodes; it is therefore chosen and moved to KNOWN group, and its value is fixed.



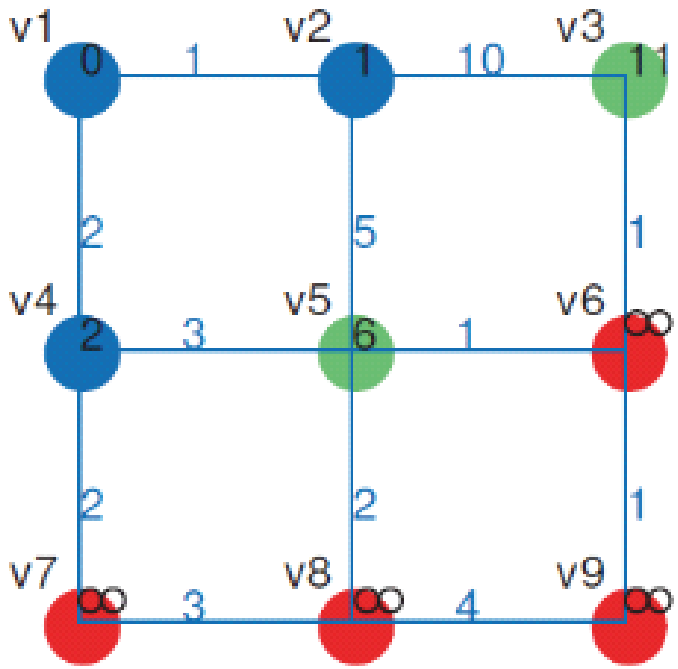
Dijkstra's Algorithm

The FAR neighbors of v_2, namely v_5 and v_3, are moved to the TRIAL group and assigned temporary distance values.



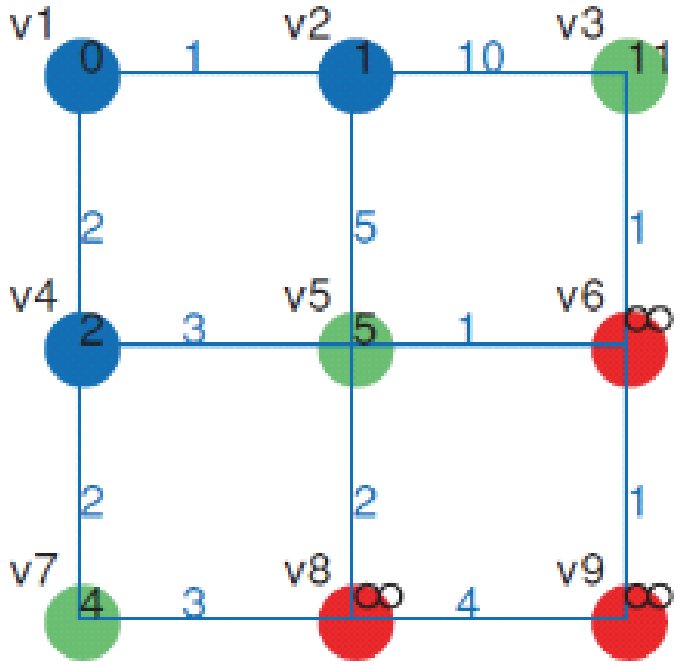
Dijkstra's Algorithm

$v4$ was the TRIAL node with smallest distance, therefore it got chosen and moved to the KNOWN group.



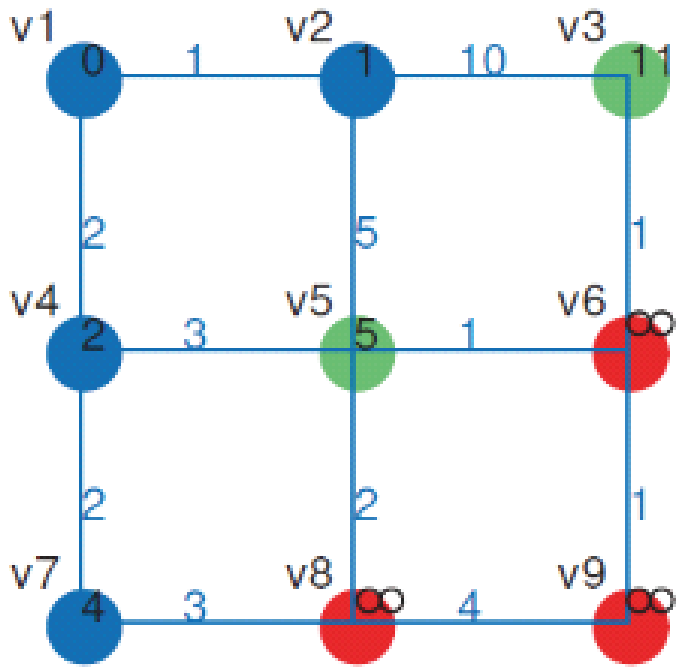
Dijkstra's Algorithm

The newly KNOWN node v_4 had one FAR neighbor, namely v_7. So v_7 got moved into the TRIAL group and assigned a temporary distance. Also, the temporary value of v_4's TRIAL neighbor v_5 got updated.



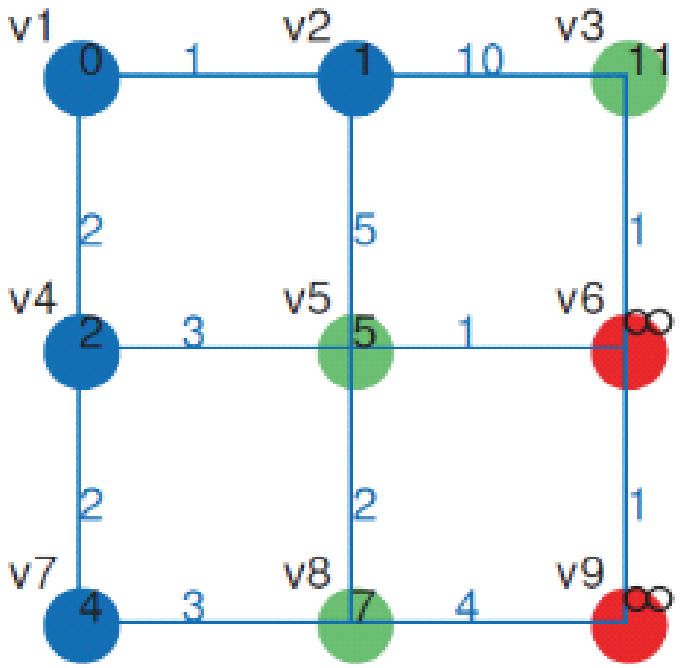
Dijkstra's Algorithm

Among the TRIAL nodes, v_7 had the smallest temporary value, so it got moved to KNOWN group and its distance is fixed.



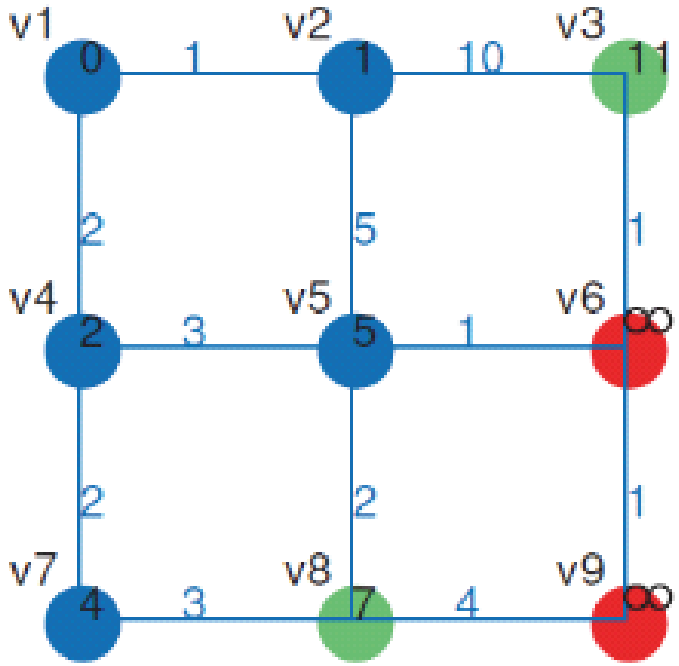
Dijkstra's Algorithm

The newly KNOWN node v_7 had a FAR neighbor v_8; so this neighbor got moved to TRIAL group and assigned a temporary value.



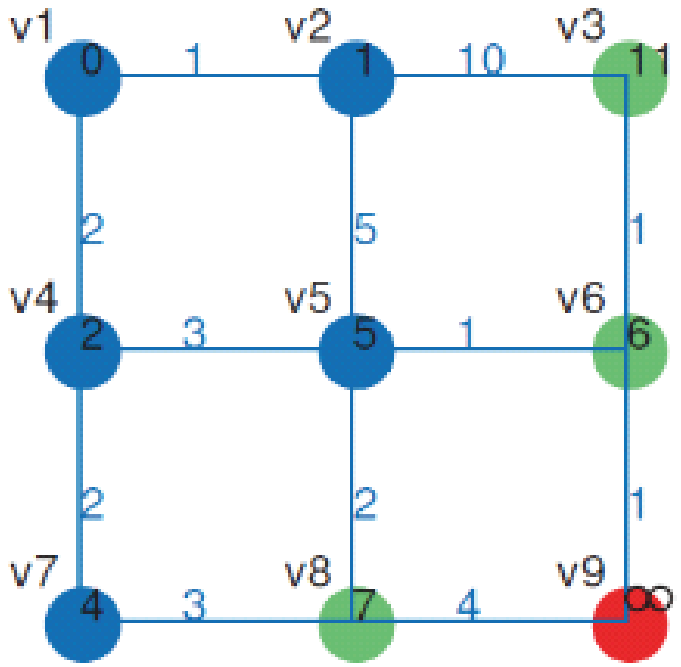
Dijkstra's Algorithm

Among the TRIAL nodes, v_5 had the smallest temporary value; so it got moved to KNOWN group, and its distance value is fixed.



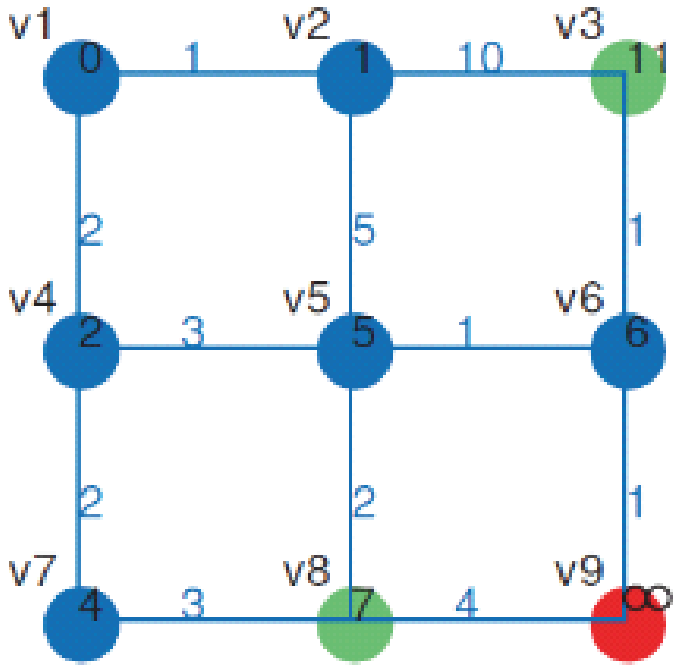
Dijkstra's Algorithm

The newly KNOWN node v_5 had a FAR neighbor v_6, which got moved to TRIAL group and assigned a temporary value; the TRIAL neighbor of v_5, namely v_8, did not need to be updated.



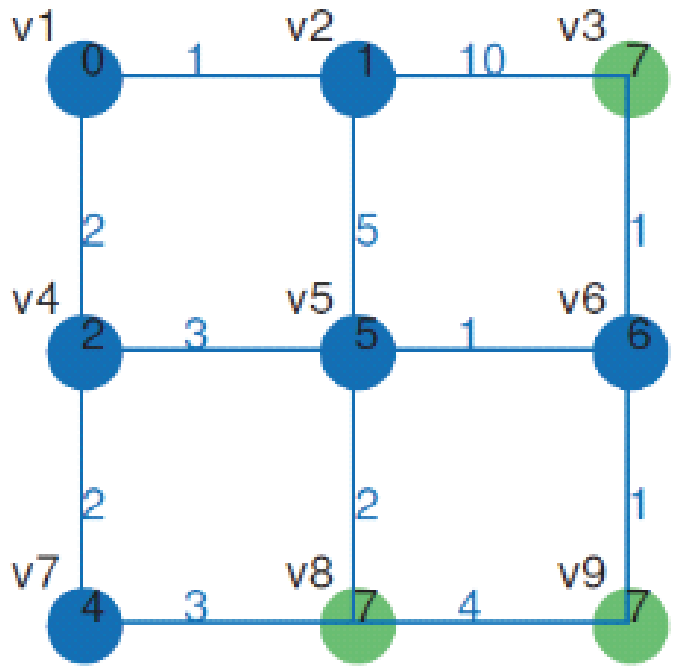
Dijkstra's Algorithm

Among the TRIAL nodes, v_6 has the smallest temp. value; so it got moved to KNOWN group and its distance is fixed.



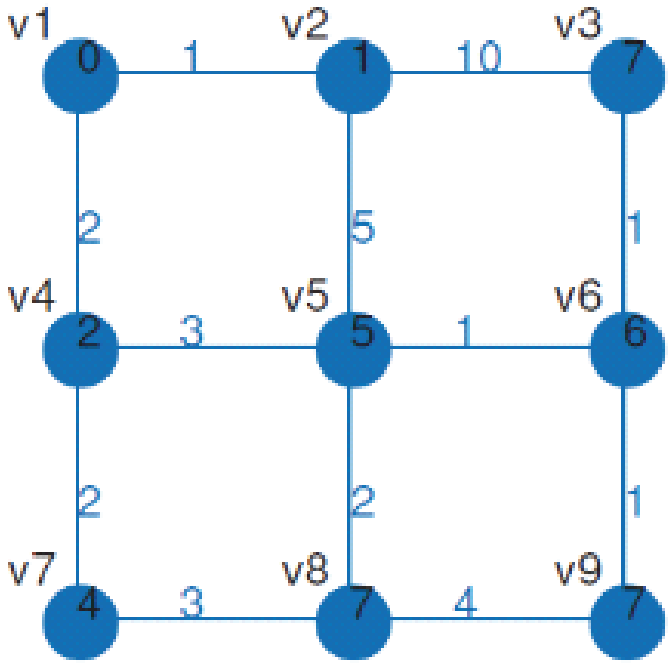
Dijkstra's Algorithm

The newly KNOWN node v_6 had a FAR neighbor v_9, which got moved to the TRIAL group and assigned a temp. value. The TRIAL neighbor v_3 of v_6 had its temp. value updated.



Dijkstra's Algorithm

Since all the remaining nodes are TRIAL nodes with minimal temp. value of 7 each, they can all be moved to KNOWN set. The algorithm is finished.



Fast Marching

- Tsitsiklis & Sethian.
- Use a discretization of

$$|\nabla u| = g$$

that gives the viscosity solution, e.g.

$$\max(\max(D_x^- u, 0), -\min(D_x^+ u, 0))^2 + \max\left(\max\left(D_y^- u, 0\right), -\min\left(D_y^+ u, 0\right)\right)^2 = g(x, y)$$

to update distance values of a grid point based on its neighbors during the course of Dijkstra's algorithm.

ALTERNATIVE: Fast marching of Osher, Tsai, Zhao, et. al.