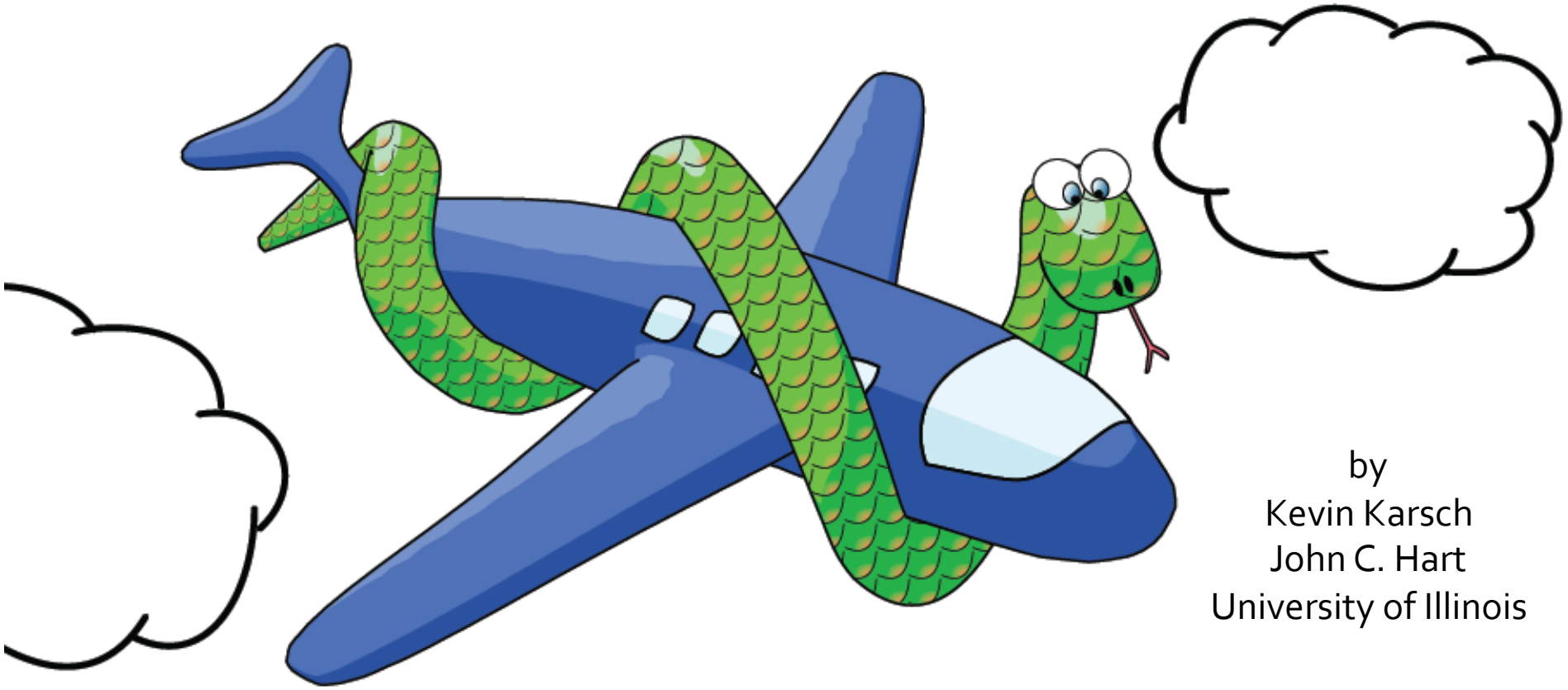
Snaxels on a Plane

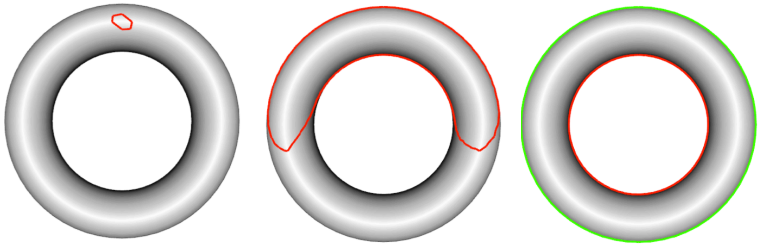


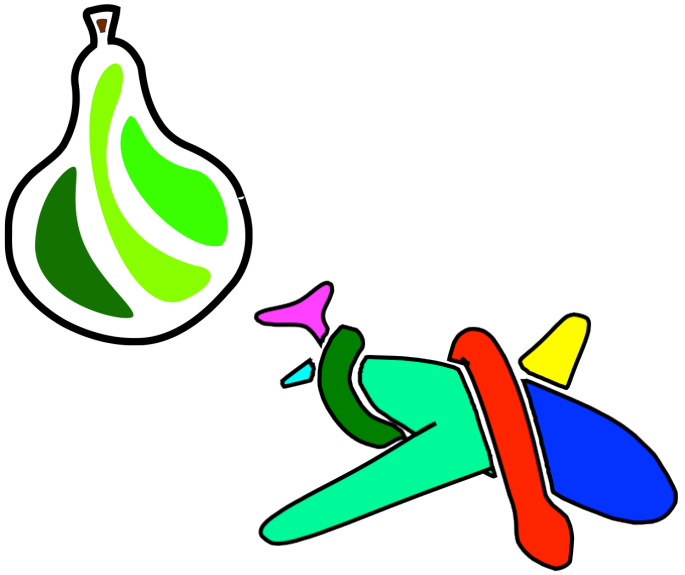
by
Kevin Karsch
John C. Hart
University of Illinois

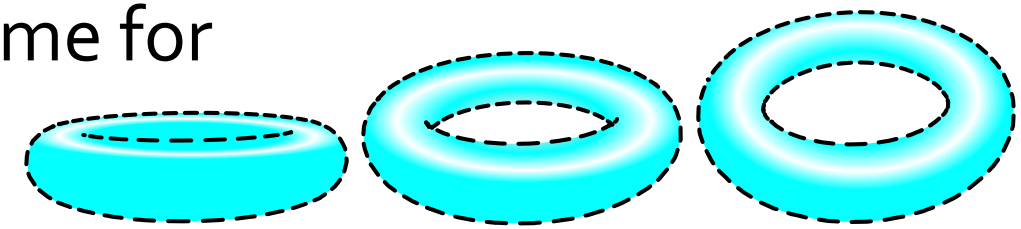
"I have had it with these multi-front snakes on this meshed facet plane!"

What can snaxels do?

- Robustly find closed contour loops on meshes
- Simplify the creation of stylized (closed region) line art
- Provide a quick and intuitive framework for generating planar maps
- Track contours over time for coherent stylization



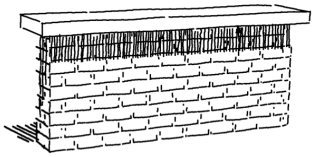


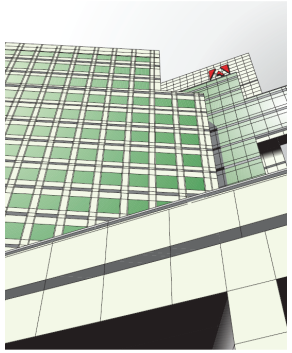


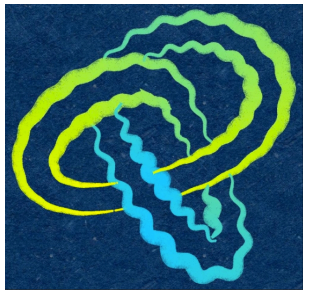
Motivation

- Extract stylistic edges from meshes
  - Real-Time [Markosian et al. 97]
  - Suggestive Contours [Decarlo et al. 03]
  - Apparent Ridges [Judd & Durand 07]

- Need to link edges into closed loops
  - Hatching [Winkenbach & Salesin 94]
  - Planar maps [Asente et al. 07]
  - Stylization [Grabli et al. 04]
  - Diffusion gradients [Orzan et al. 08]

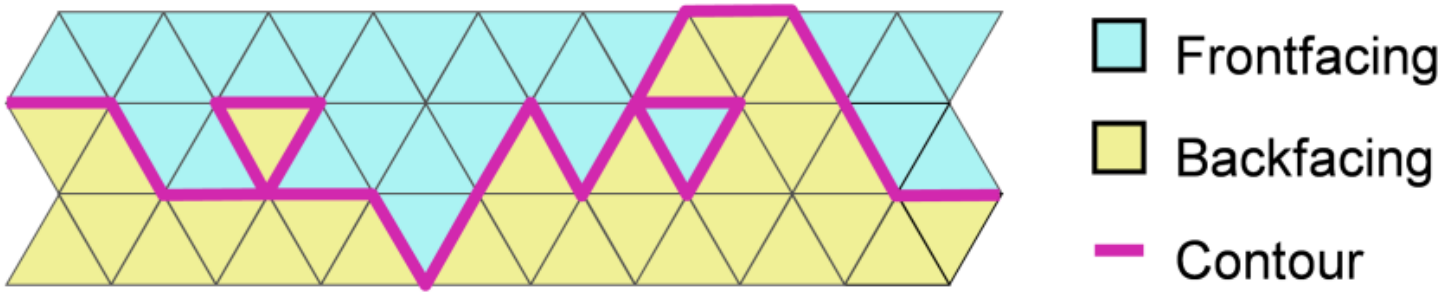






Motivation

- Tricky to assemble edges into closed loops



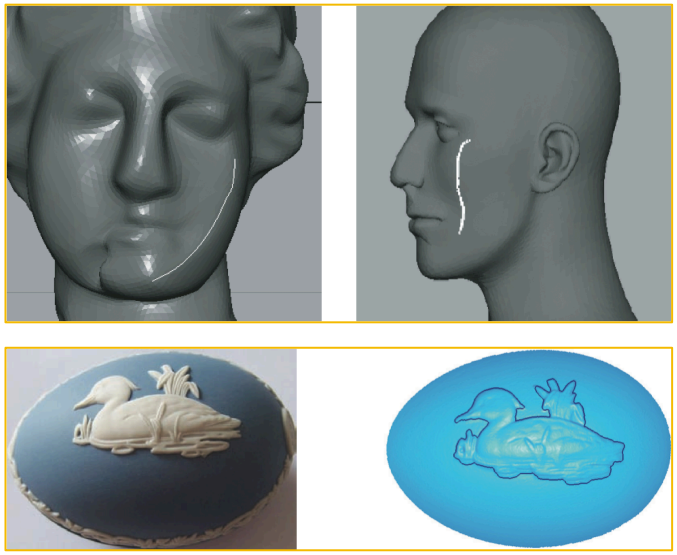
From Rusinkiewicz's notes on "Algorithms for Extracting Lines"

- Robust contour loop/planar map extraction is "complicated"
  - Switchbacks, slivers, etc.
  - Robustness and efficiency [Asente et al. 2007]
  - Heavyweight packages (CGAL) [Eisemann et al. 2008]
  - Entire appendix on tolerancing [Eisemann et al. 2009]

Solution: Active Surface Contours

- Represent closed loop illustration contours with closed loop surface contours

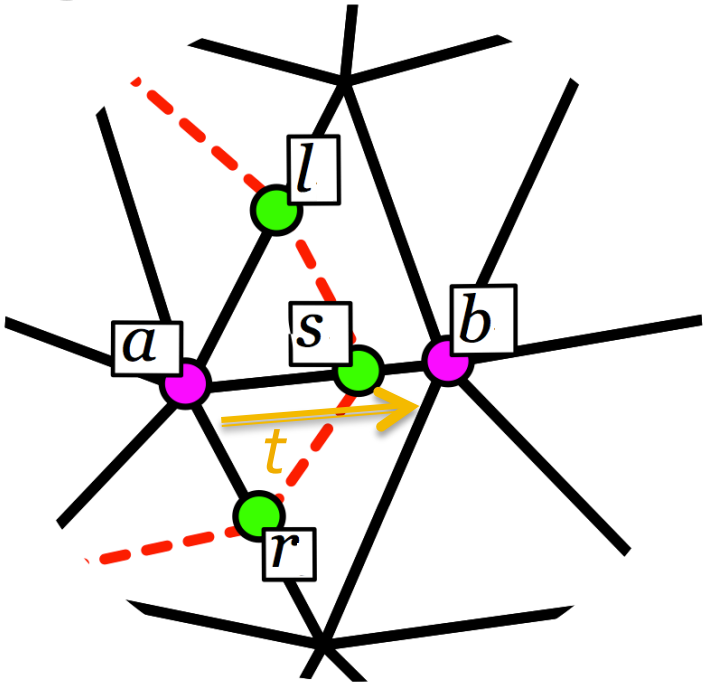
- Meshed surface "snakes" [Kass et al. 88]
  - Segmenting mesh features [Jung & Kim 04]
  - Detecting relief on meshes [Lui et al. 06]



- Easy robust implementation: "snaxels" [Bischoff et al. 05]

Snaxels [Bischoff et al. 05]

- Active contour is a chain of vertices called "snaxels"
- Each snaxel must lie on a unique mesh edge
- Each snaxel connects to neighbors across face
- Each ring of connected snaxels is called a *front*

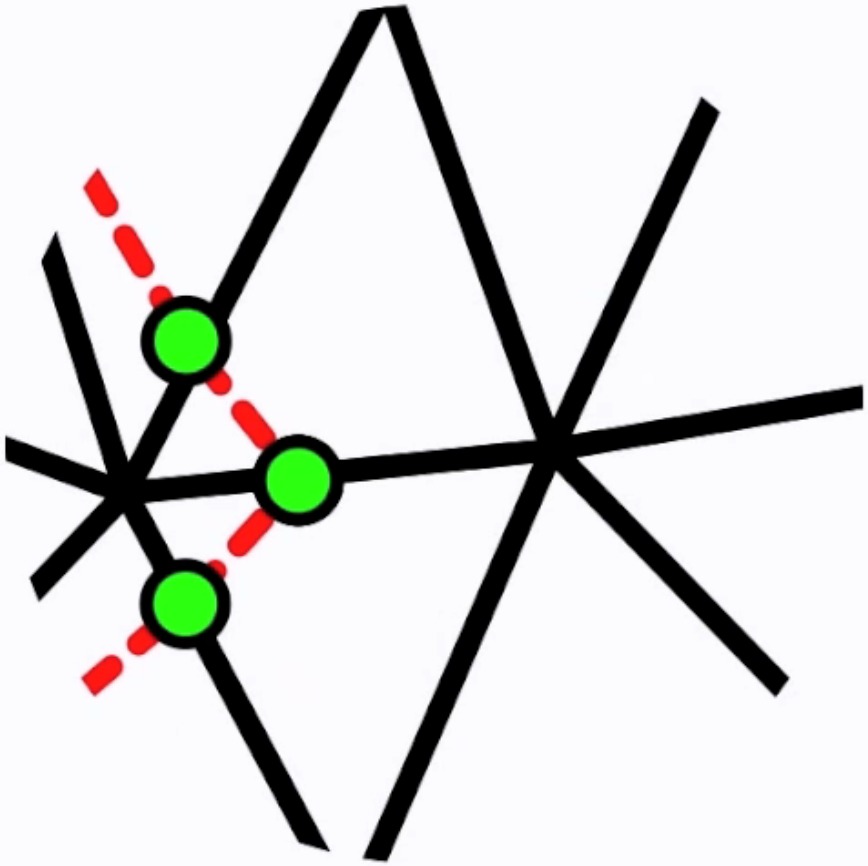


Snaxel "s" lies on the edge (a,b) parameterized by t ∈[0,1] with neighbors l and r in its contour

$s = (a, b, t, l, r)$

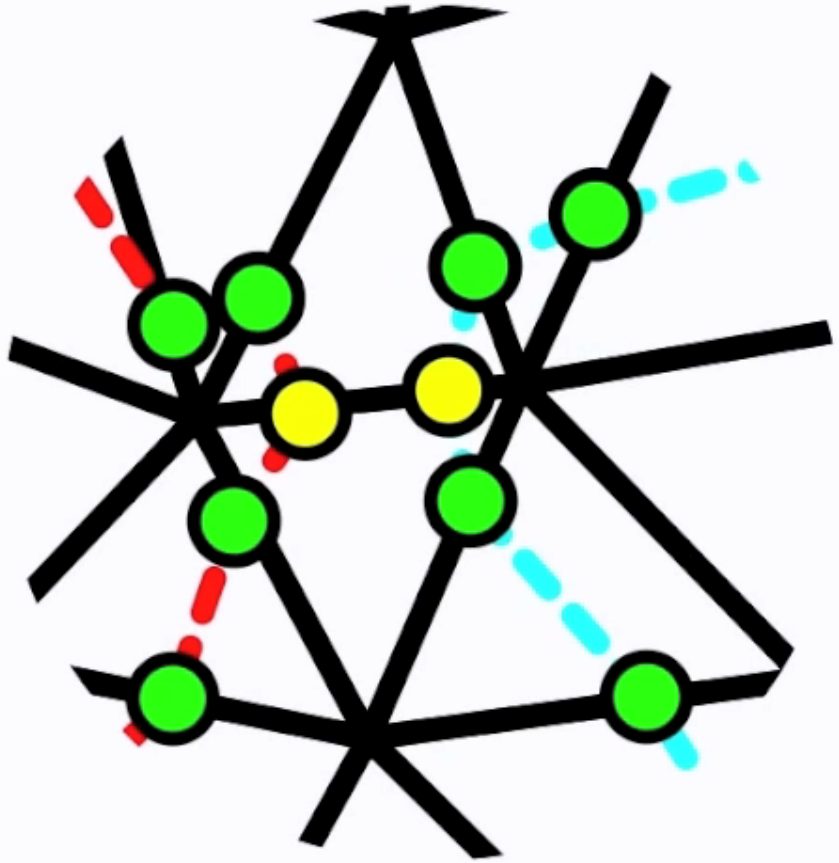
Implementation

- *Fan-out* when a snaxel reaches a vertex



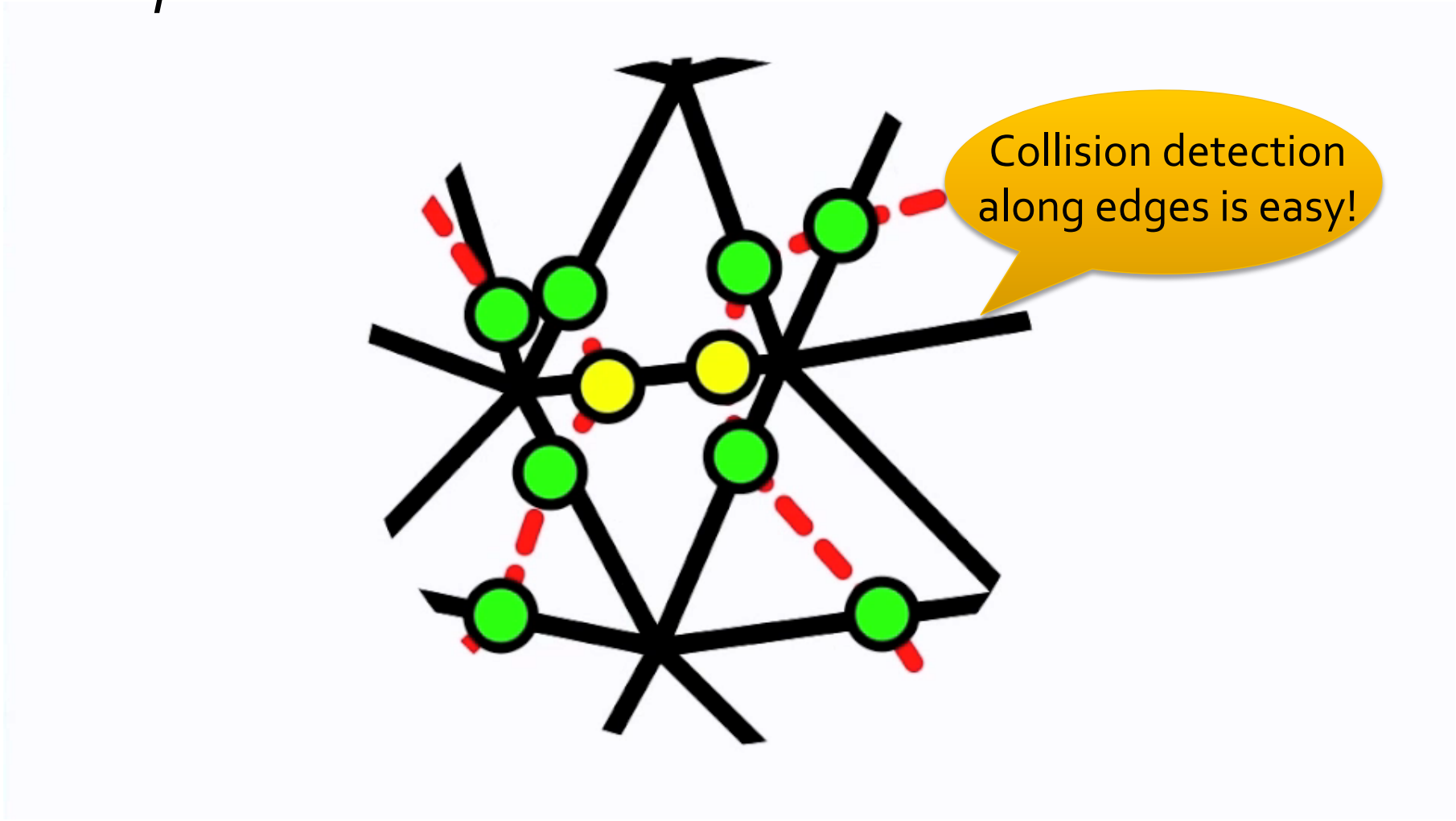
Implementation

- *Merge* when two fronts collide



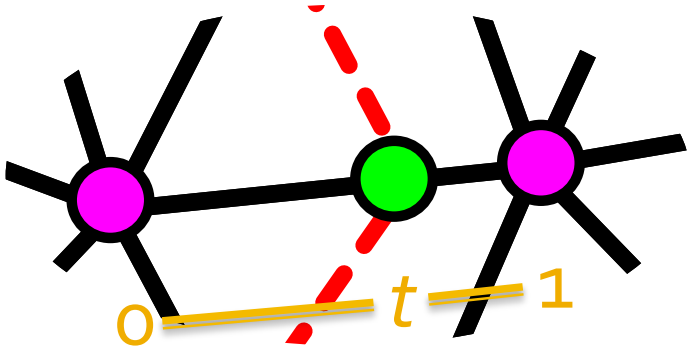
Implementation

- *Split* when a front intersects itself



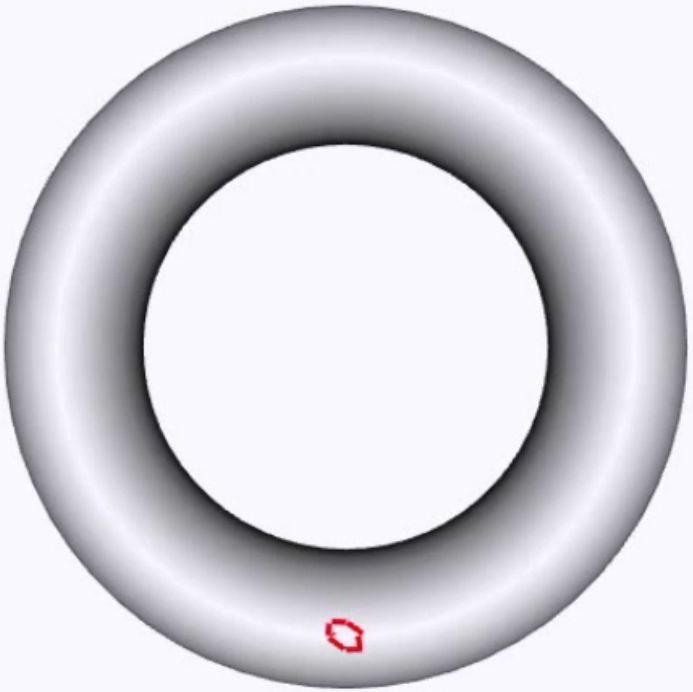
Implementation

- Snaxels minimize f



$$t_i \leftarrow t_i - \frac{\Delta t}{||b_i - a_i||} f(s_i)$$

- Energy is based on the desired contour



- Visual contour generator

$$f(s_i) = N_i \cdot V_i$$

Isophotes

- We can also use snaxels to simulate diffuse lighting
  - $f(s_i) = \min(N_i \cdot L_i - k, N_i \cdot V_i)$
  - *k* is the isovalue of an isophote from a point light

Pseudocode

$$\mathbb{S} \leftarrow \textsc{initializeSnaxels}()$$
$$\text{while } \textsc{energy}(\mathbb{S}) > \epsilon$$
$$\quad \text{for } s_i \in \mathbb{S}$$
$$\quad\quad s_i^p \leftarrow s_i^p - \frac{\Delta t}{||s_i^{edge}||} f(s_i)$$
$$\quad\quad \mathbb{S} \leftarrow \textsc{handleCollisions}(s_i, \mathbb{S})$$
$$\textsc{processForNPR}(\mathbb{S})$$

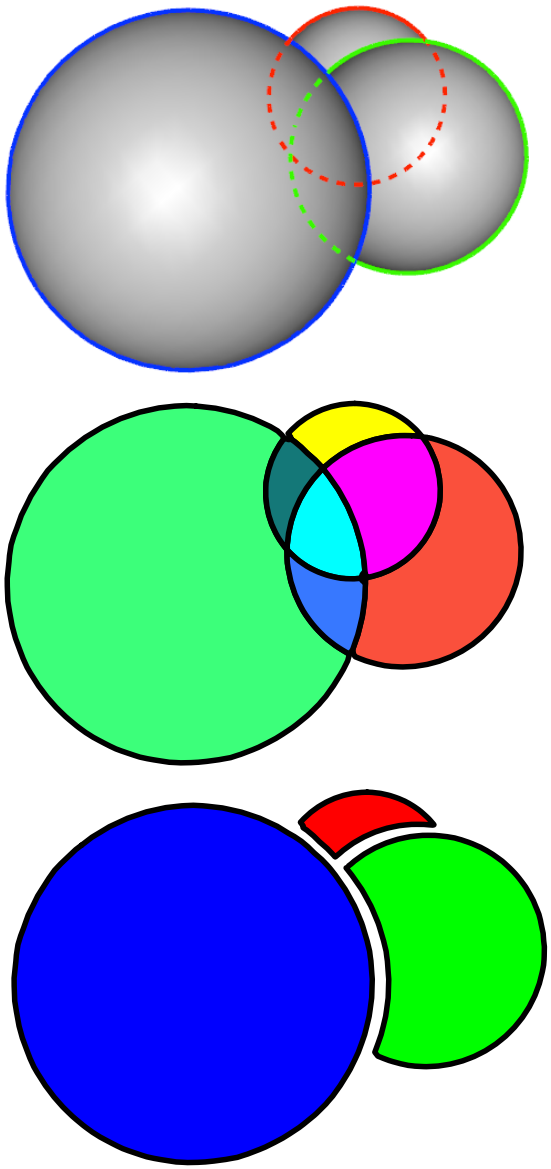
Update position based on $f(\cdot)$

Fan-out, merge, split if necessary

Stylize contours

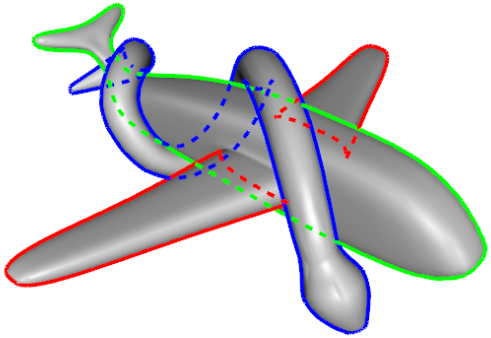
Planar Maps

- Planar graphs, i.e. edges intersect only at vertices
- Planar map organizes overlapping geometric elements into separate planar regions
- Planar map organizes 3-D projection into regions based on quantitative visibility
- NPR primarily interested in visible portions of the planar map

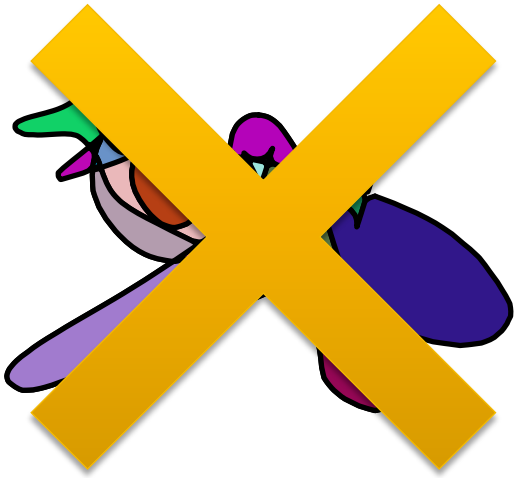


Planar Map Generation

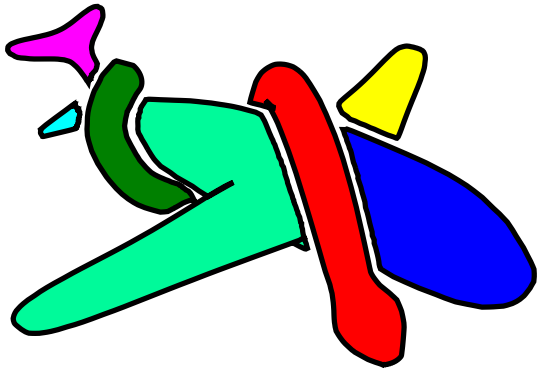
- Snaxels facilitate intuitive and easy implementation
- Avoids expensive processing and imprecise boundaries



Meshed geometry



Post-processed



Snaxel-defined

Planar Map Generation

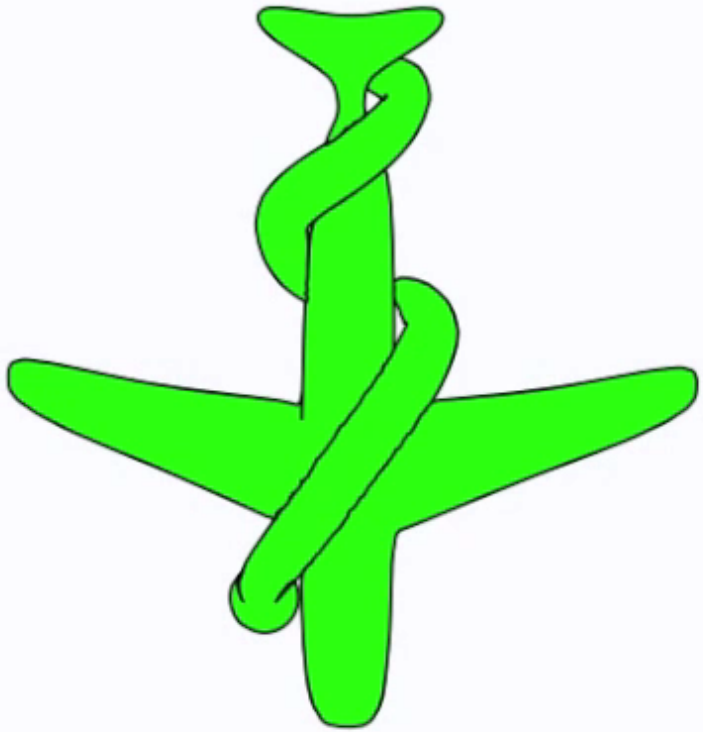
- Planar map generated by a simple addition to the existing snaxel framework
  - Snaxel fronts cannot overlap in view space
  - Priority given to fronts closer to camera (closer fronts push farther fronts back)

$$f_{pmap}(s_i) = \delta(s_i) f(s_i), \text{where}$$

$$\delta(s_i) = \begin{cases} 1 & \text{if } s_i \text{ is within a closer front} \\ -1 & \text{otherwise} \end{cases}$$

Planar Map Generation

$$f_{pmap}(s_i) = \delta(s_i) f(s_i)$$

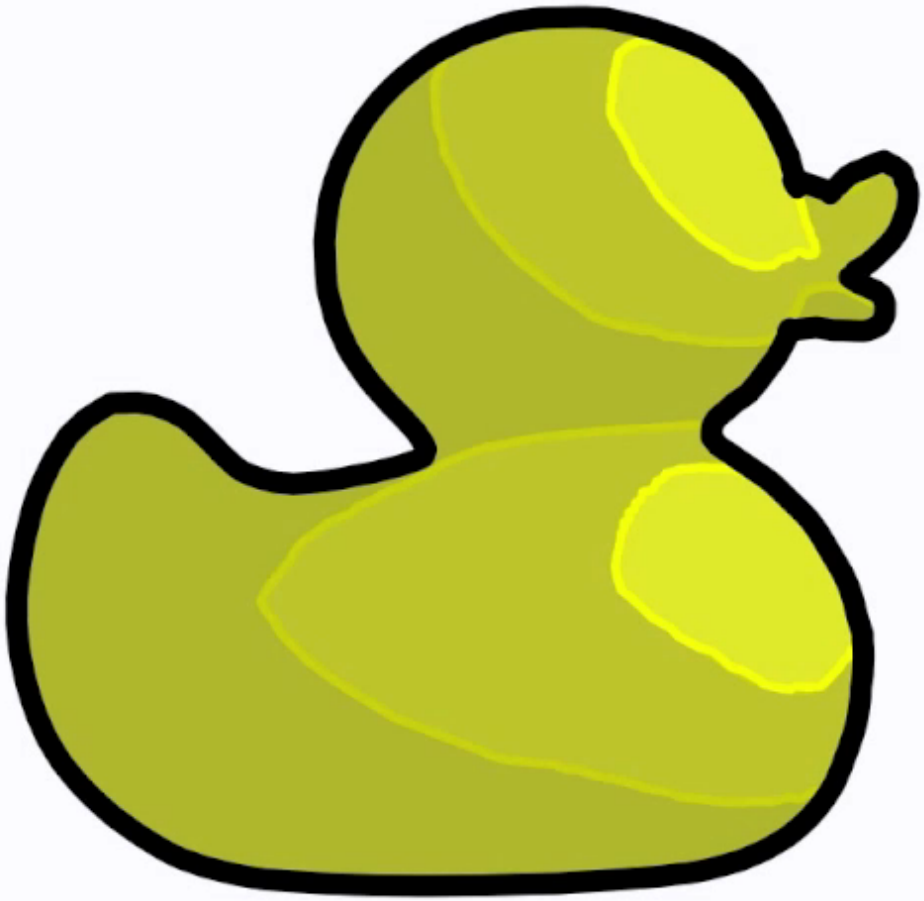


Planar Map Generation

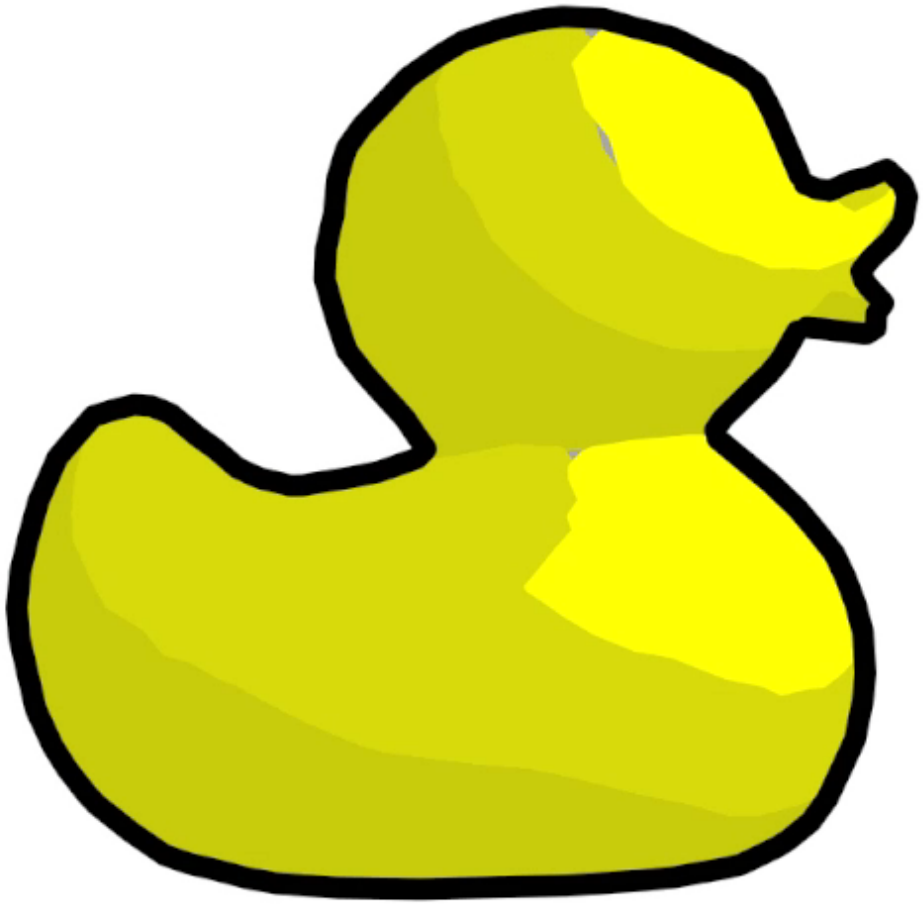
$$f_{pmap}(s_i) = \delta(s_i) f(s_i)$$

Planar Map Generation

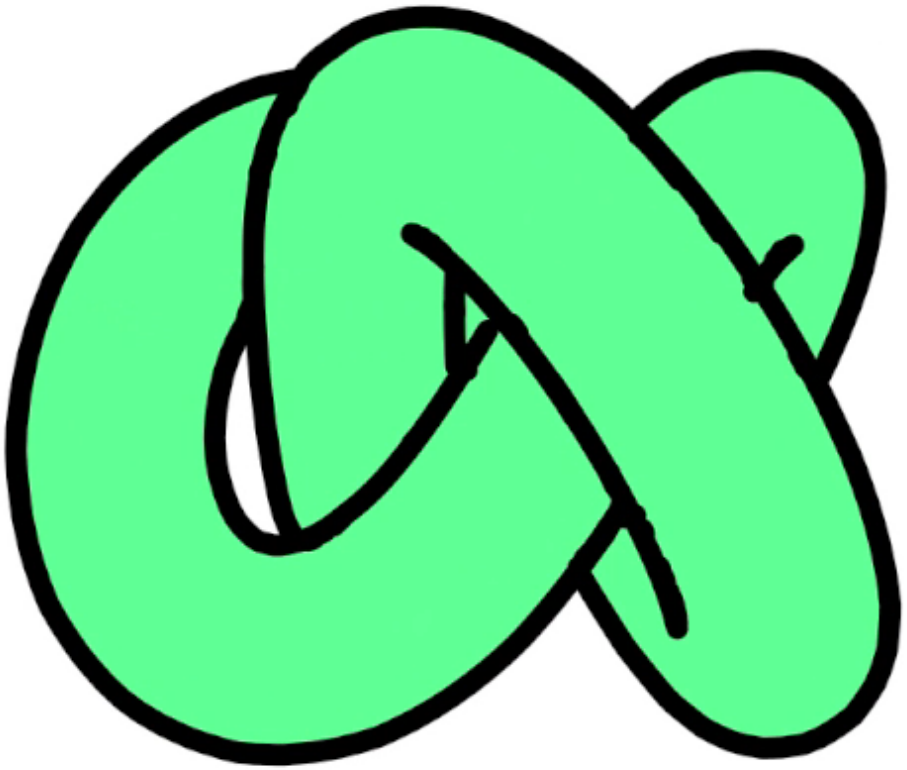
$$f_{pmap}(s_i) = \delta(s_i) f(s_i)$$



Stylized planar maps

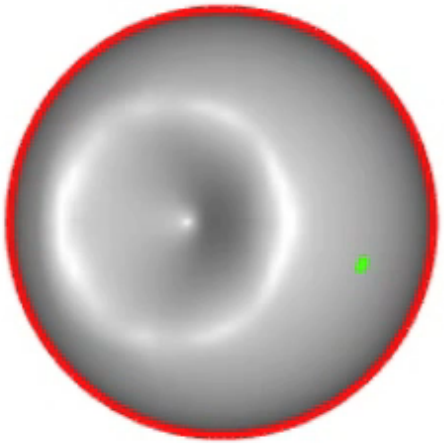


Stylized planar maps



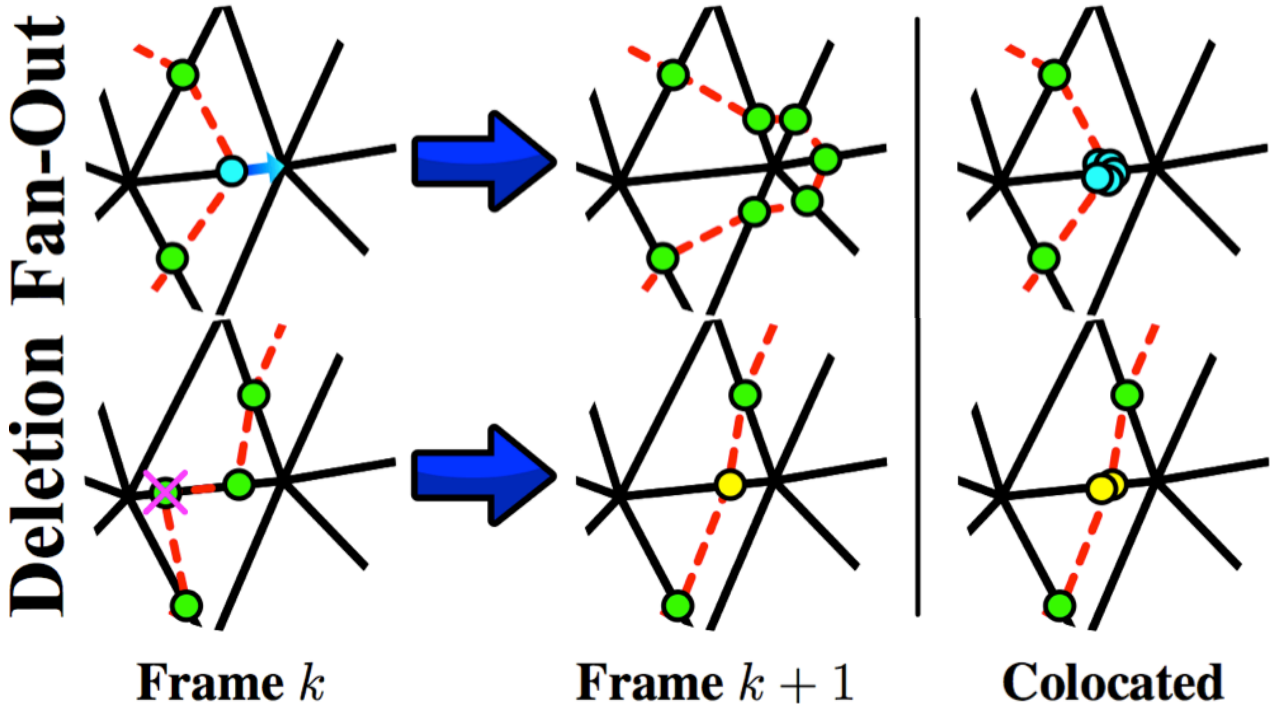
Animated Contours

- Snaxel fronts not only grow into desired surface contours…
- They also track evolving contours as a mesh moves, rotates or changes shape
  - No re-initialization
  - Robust to topology changes



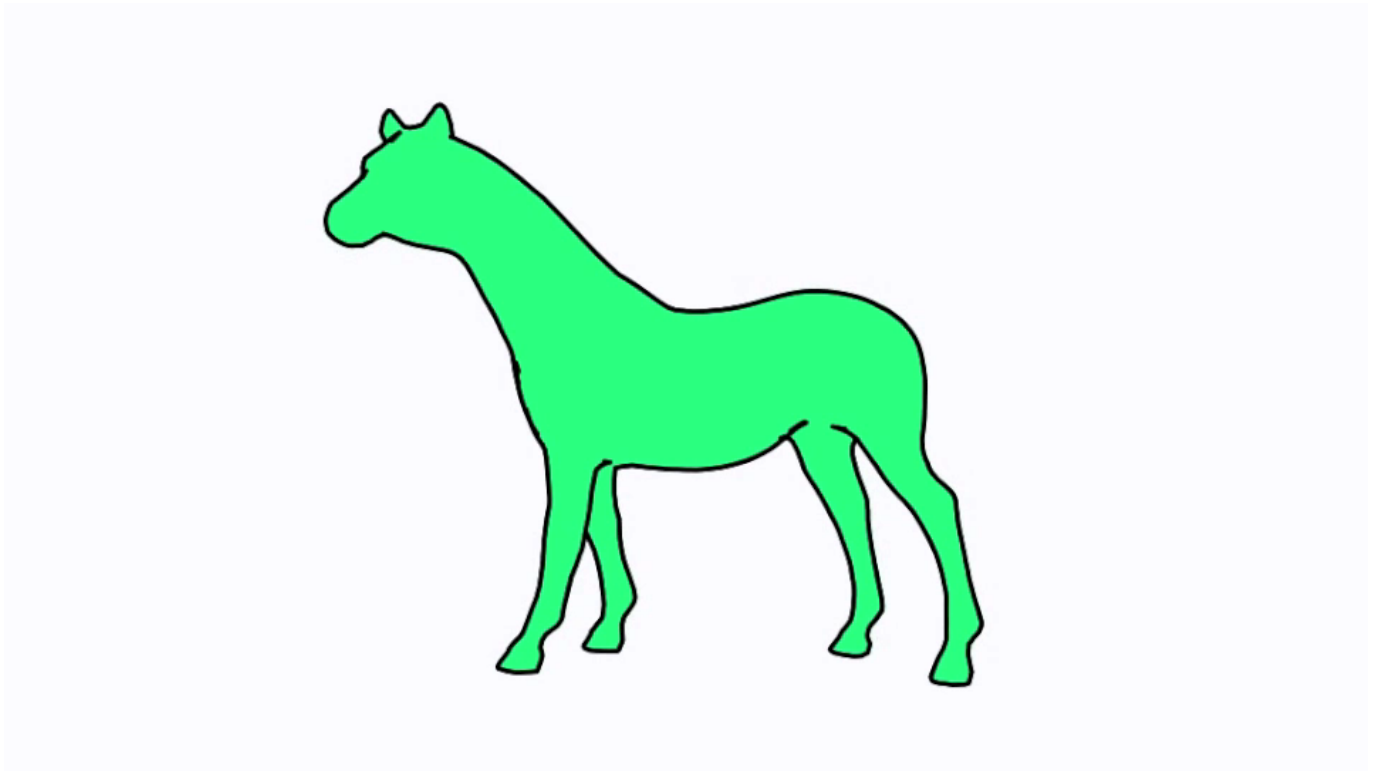
Tracking contours

- Explicit contours provide direct correspondence across frames
- For 1-1 correspondences, each contour must have same number of vertices throughout the animation
  - Changes in contour topology handled by adding colocated snaxels



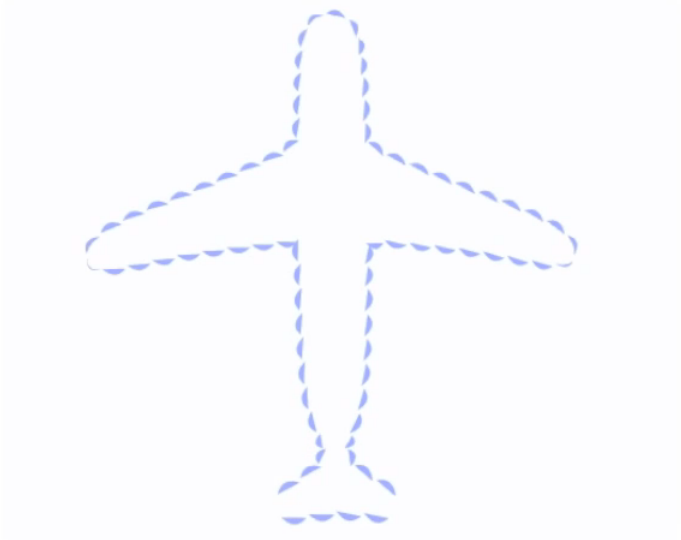
Tracking contours

- Temporal correspondences can be used to
  - Convert mesh animations to animated SVG



Tracking contours

- Temporal correspondences can be used to
  - Convert mesh animations to animated SVG
  - Temporally coherent silhouettes
    - Similar to [Kalnins et al. 2003], but with explicit, closed loop contours



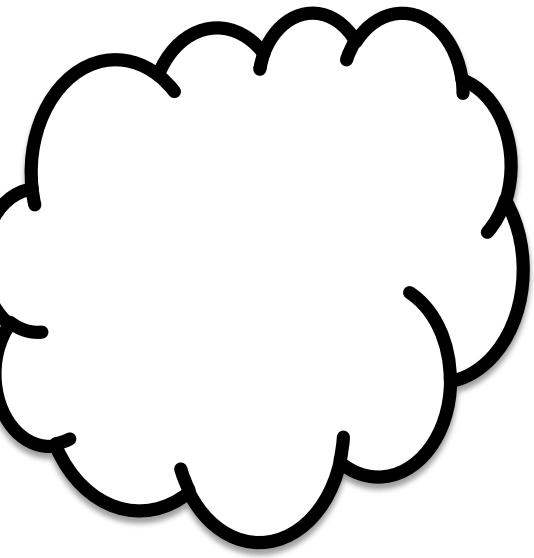
Contributions

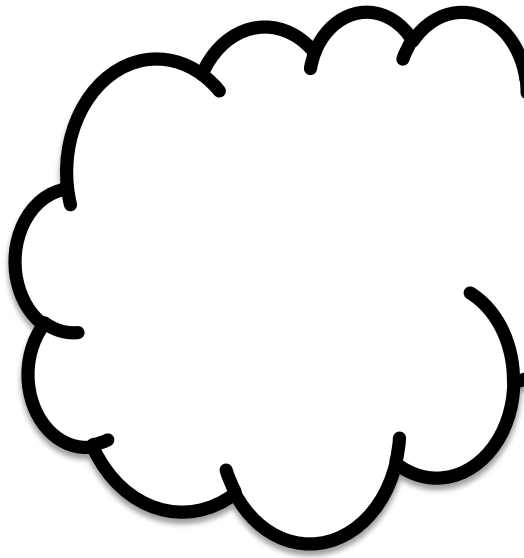
- Snaxels for NPR
  - Supports many useful illustrative techniques
  - Easy to implement
  - Highly customizable
- Intuitive planar map scheme
  - Avoids large libraries and complex algorithms
- Efficient contour tracking
  - Useful for 2D SVG animation, coherent stylization

Future work

- Further investigate energy function
  - Curvature/shape constraints
  - Other planar map schemes
- Parallel implementation
  - Fan-out/collision events pose challenges for efficient parallelization
- Bezier contour tracking
  - Simplify/enhance curve animations with higher order curves

Questions?





Thanks!