Non-Photorealistic Rendering

Stroke-Based Rendering

Tobias Isenberg

Overview

- introduction and overview
- optimization strategies
- greedy algorithms

Stroke-Based Rendering

Introduction and Overview

Introduction

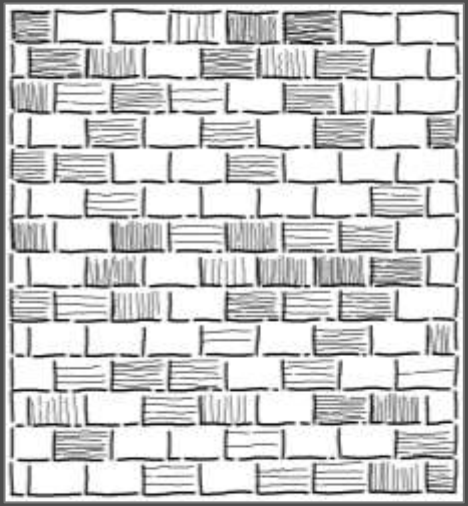
- stroke as the main primitive
  - (semi-)automatic placement of discrete elements
  - use of objective function to measure placement quality
  - typical goal: represent an image or a 3D shape
  - result: abstracted version of the source image/shape

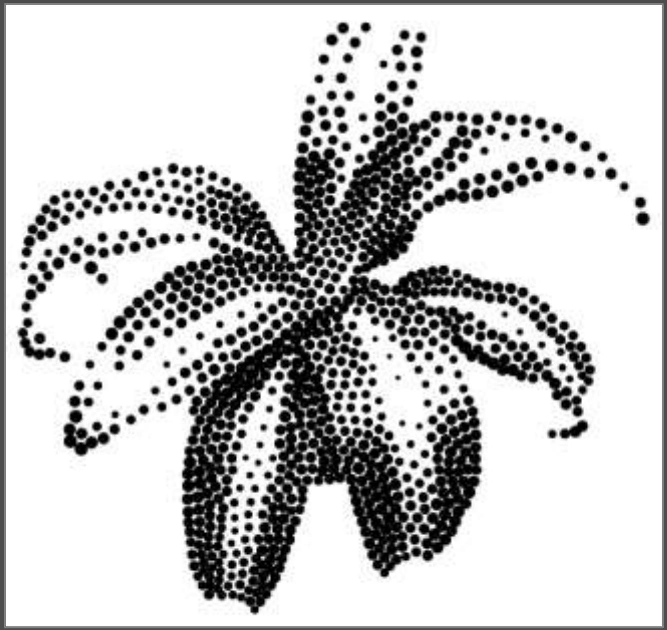
Introduction

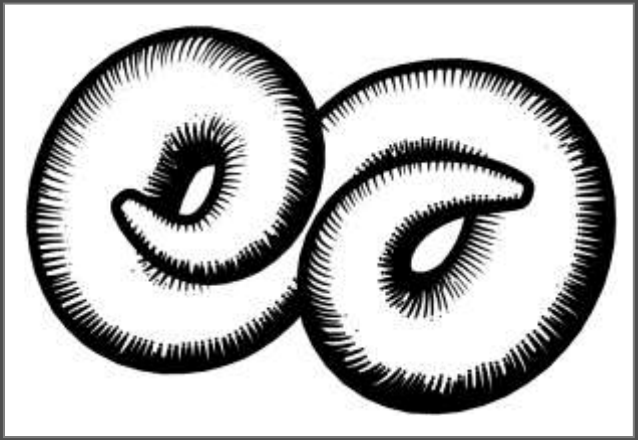
- control of stroke placement
  - automatic
  - semi-automatic
  - interactive
- human control at all levels
  - decision on the stroke set
  - decision on the source image
  - parameters of objective function
  - semi-automatic stroke placement
  - interactive stroke placement

Introduction

- some pen-and-ink techniques are stroke-based techniques
  - e.g., prioritized stroke textures
  - e.g., relaxation-based stippling
  - e.g., hatching rendering

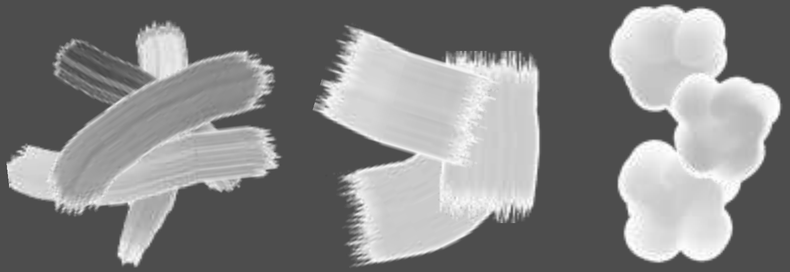






Definitions

- necessary elements for stroke-based rendering:
  - canvas: background color/texture/object
  - ordered list of strokes with parameterization, to be rendered (alpha-blended) onto the background



  - SBR energy function: measurement of image quality

    $E: I \rightarrow R$; I = set of possible images, R = real numbers

Energy Function E(I)

- measures how closely input image is matched
- also encodes trade-offs
  - e.g., abstraction by enforcing larger strokes
  - otherwise very many (m×n) tiny paint strokes (pixels)
- example for an SBR energy function

$$E(I) = E_{match}(I) + w_{abs} E_{abs}(I)$$

$$E_{match} = \sum_{(x,y) \in I} \left\| I(x,y) - S(x,y) \right\|^2$$

$$E_{abs}(I) = \text{number of strokes in } I$$

  - w_{abs} is a factor to parameterize the influence of E_{abs}
  - S is the source image

Energy Function E(I)

$$E(I) = E_{match}(I) + w_{abs} E_{abs}(I)$$

- energy function E(I) is minimized
  - inspired by physics where minimum energy configurations for particle sets are sought
- depending on w_{abs} :
  - E_{match} has more influence (small w_{abs}), then the image looks more like the source image, and the strokes are less evident → **realism**
  - E_{abs} has more influence (large w_{abs}), then the image looks less like the source image, and the strokes are more evident → **abstraction**

Stroke-Based Rendering Strategies

- optimization strategies
  - Voronoi algorithms: Lloyd’s method and variations
  - trial-and-error algorithms

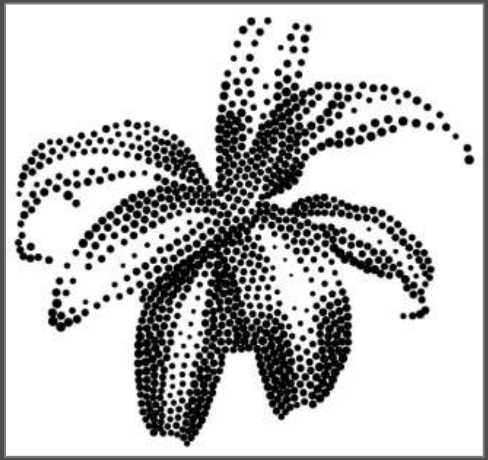
- greedy algorithms
  - rendering in one go

Stroke-Based Rendering

Optimization Strategies

Optimization Strategies for SBR

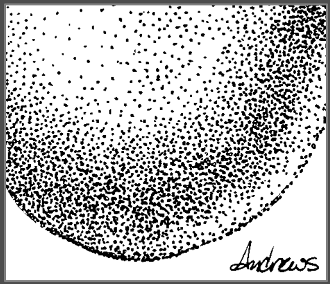
- goal: try to optimize (minimize) energy function E(I)
- two approaches:
  – Voronoi algorithms: Lloyd's method and variations
  → directed iterative optimization of a complete stroke configuration



  – trial-and-error algorithms: proposing of image changes and accepting or rejecting these
  → undirected iterative optimization using a growing set of strokes

Voronoi Algorithms

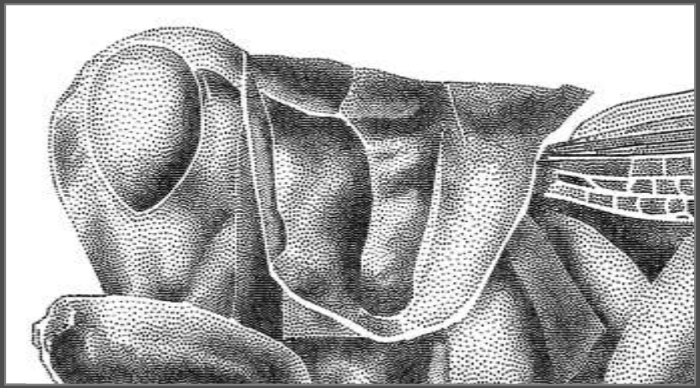
- general class of applications
  - images that contain many non-overlapping strokes
  - only stroke density is constrained
- examples
  - stippling: small black dots
  - mosaics: small colored pieces



- general idea
  - use efficient algorithms from computational geometry
  - place elements evenly, no overlaps
  - use GPU to speed up the process
- note: energy function defined as density, not tone

Lloyd’s Method: Brief Recap

- iterative optimization technique
- minimize energy function describing the distance between points and their Voronoi region centroids
- estimation of Voronoi regions can be done on GPU
- previously seen two techniques
  - interactive techniques using brushes
  - automatic technique with weighted Voronoi regions

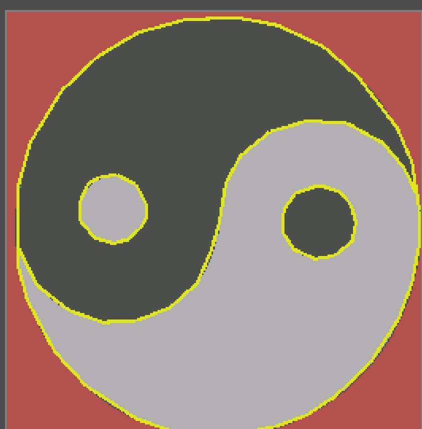
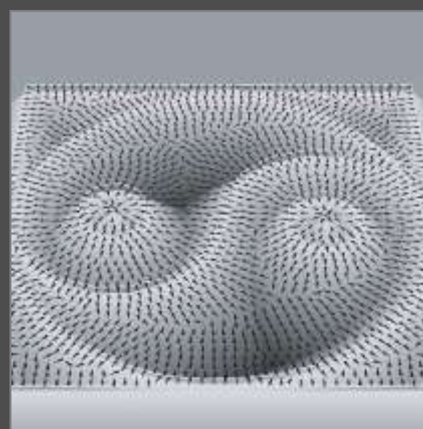


Lloyd’s Method to Create Mosaics

- mosaics: images made out of small, ca. quadratic, uniformly colored tiles
- tiles are aligned to emphasize the features in the image
- use Lloyd’s method to create shapes
- need input data:
  - image to be represented
  - edge features of the image

Lloyd's Method to Create Mosaics

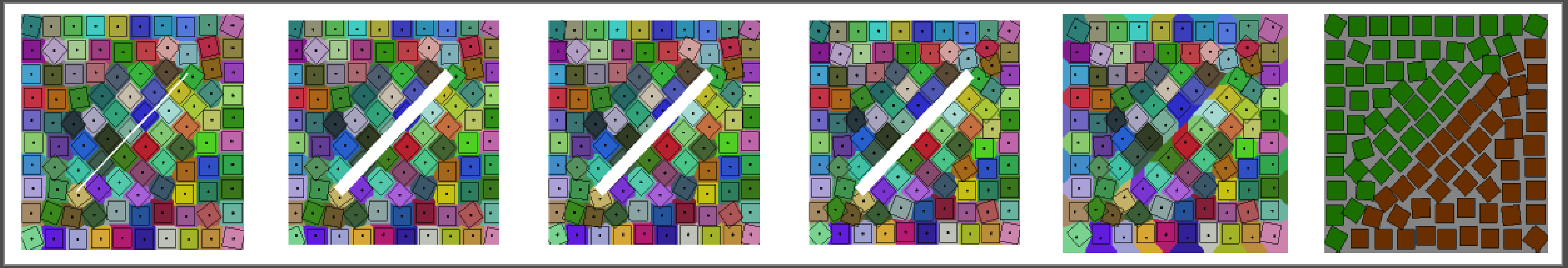
- modifications to Lloyd's method:
  - use of Manhattan distance to obtain rectangular tiles
  - use an underlying vector field to guide the computation of this metric: derived from the image's edge information

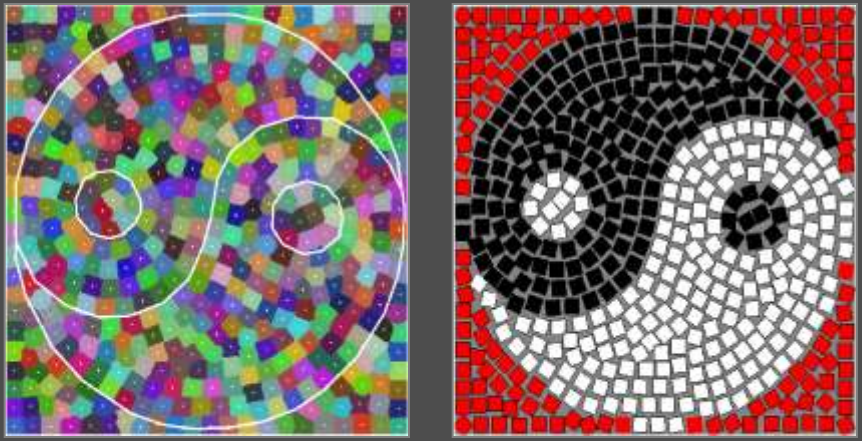
  - computing the Voronoi diagrams through rendering: pyramids at point positions with ID color, orthographic
  - results in discrete Voronoi diagram

Lloyd's Method to Create Mosaics

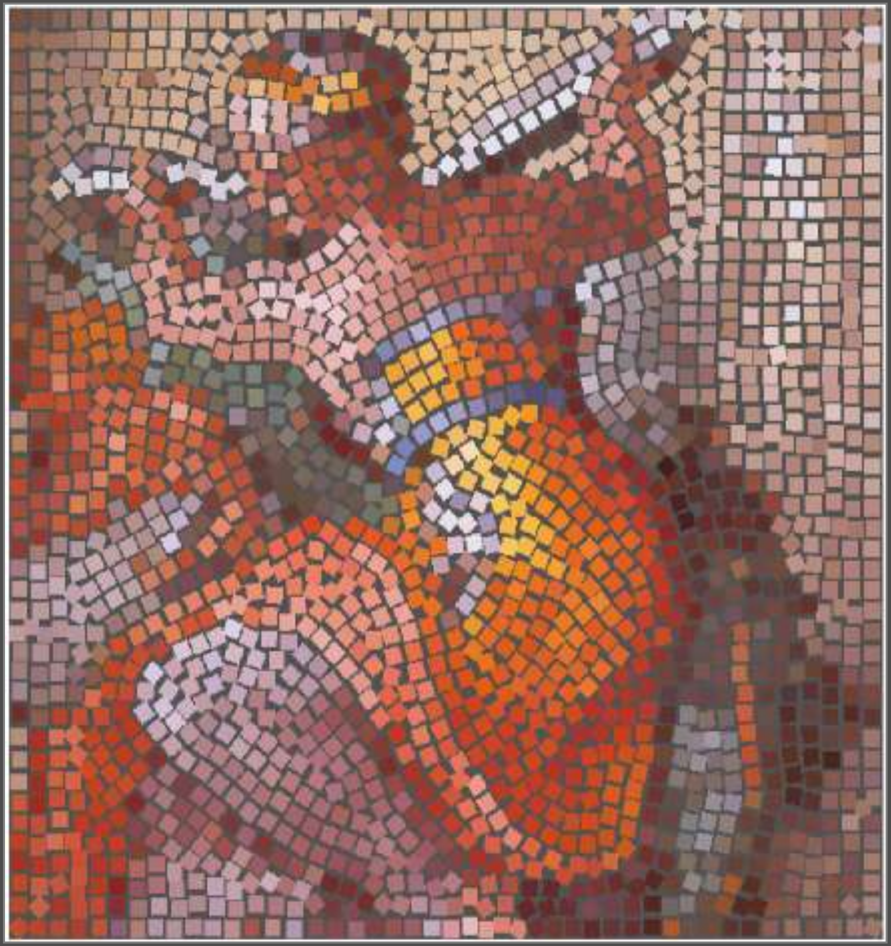
- extension of the basic technique
  - avoiding an edge by affecting the centroid computation: rendering non-ID color over Voronoi diagram affects the centroid computation so that eventually tiles avoid it



  - result: rendering of image edges included, tiles avoid features

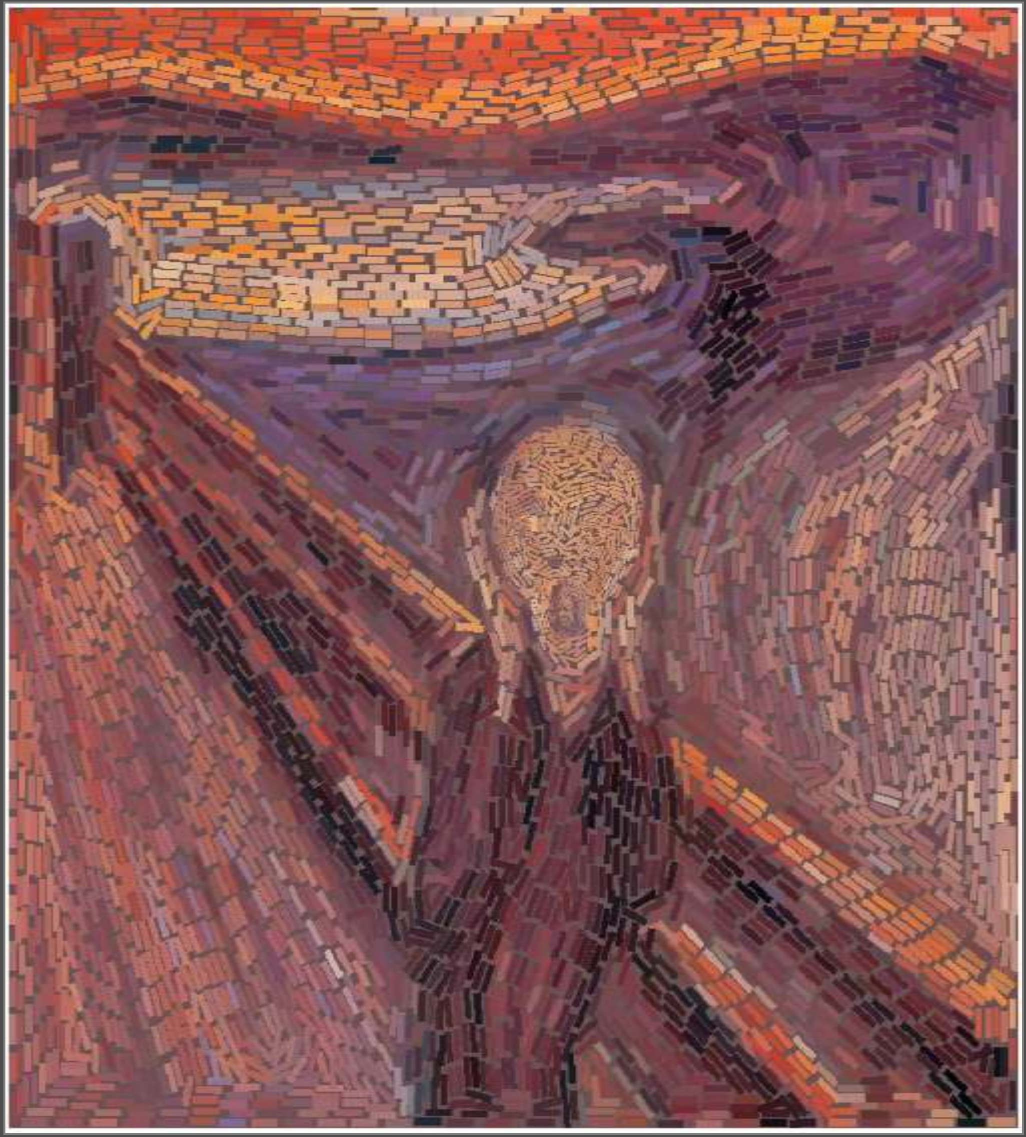


Lloyd's Method to Create Mosaics



Lloyd's Method to Create Mosaics

Lloyd's Method to Create Mosaics



Trial-and-Error Algorithms

- problem with Voronoi-based techniques: they match by distribution of elements on the image
  $\rightarrow$ only one parameter
  $\rightarrow$ not good for color

- thus, new approach: trial-and-error
  – propose changes to a current version of the image
  – if the change minimizes the energy, accept it
  – if the change does not minimize it, reject it
  – repeat until result is satisfying or for a certain # of times

Trial-and-Error: First Example

- approach based on stroke parameter perturbation
  - distribute a number of strokes on the canvas
  - strokes read color from background image
  - in each iteration step, randomly perturb the parameters of a stroke
  - check if the sum-of-squares difference to the source image is reduced
  - repeat until satisfied with the result

Trial-and-Error: Painterly Rendering

- goal: concise painting that matches source image coarsely and cover the image with few paint strokes
  - strokes defined by brush radius and list of control points
  - energy function: $E(I) = E_{match}(I) + E_{nstr}(I) + E_{\text{cov}}(I)$

$$E_{match} = \sum_{p \in I} w_{match}(p) \left\| I(p) - S(p) \right\|^2$$

$$E_{nstr}(I) = w_{nstr}(\text{number of strokes in } I)$$

$$E_{\text{cov}}(I) = w_{\text{cov}}(\text{number of empty pixels in } I)$$

  - $w_{match}(p)$ allows to control detail in the painting, can be initialized using edges extracted from an image
  - w_{cov} can be used to force all pixels to be filled
  - procedure: add strokes with trial-and-error until satisfied

Trial-and-Error: Painterly Rendering

source image

interactively painted weight image w_{app}

result for one set of weights

result for another set of weights

Trial-and-Error Algorithms

- problems with trial-and-error algorithms
  - algorithms need to be well-designed to converge to low-energy configuration/result
  - the search is not directed
  - thus, implementations may need to run for a long time to return the desired results

- thus, a more goal-oriented type of techniques are necessary and required:

  greedy algorithms → next

Stroke-Based Rendering

Greedy Algorithms

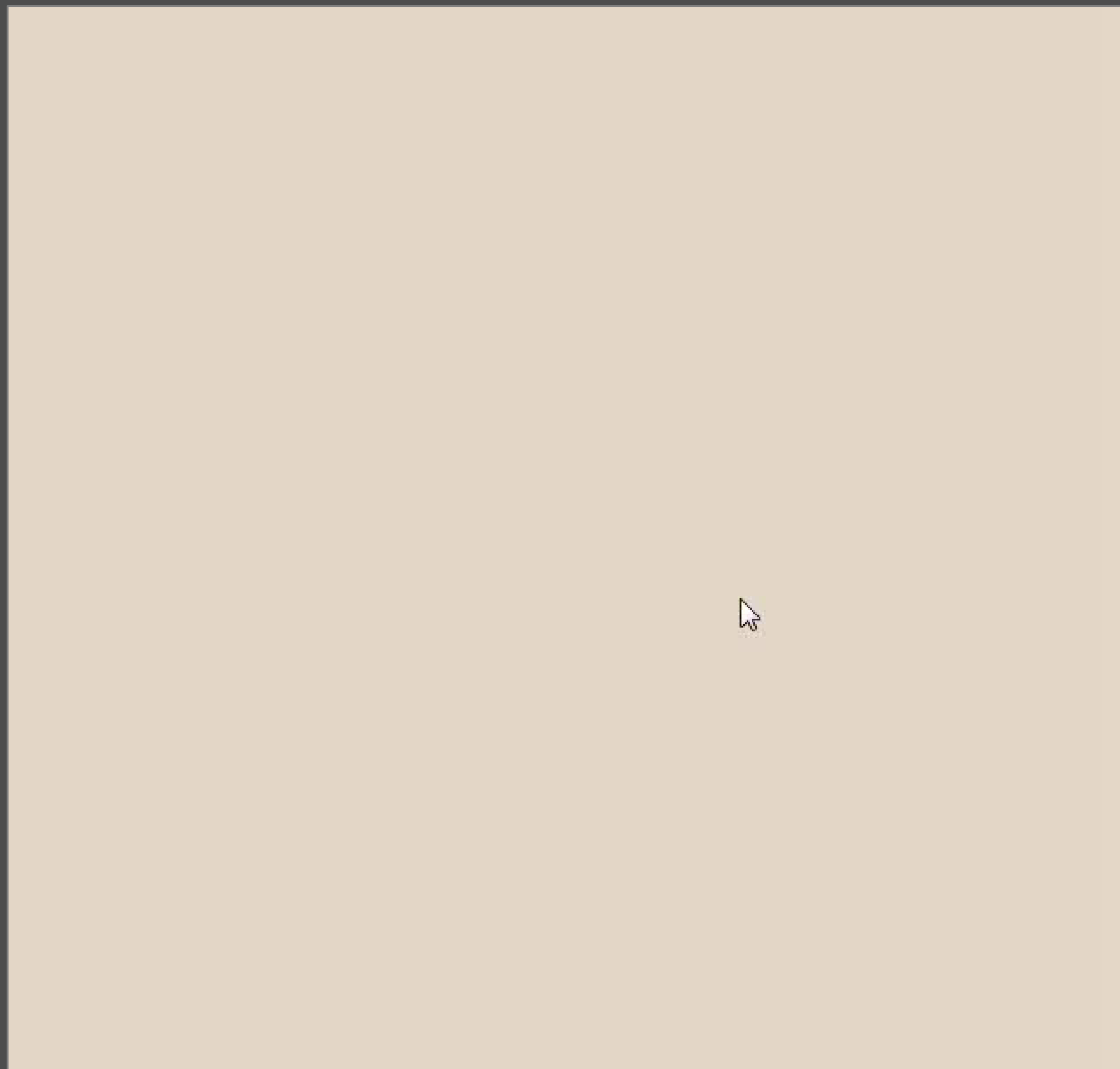
Greedy Algorithms

- strokes added to image in a single pass
- use well-designed heuristic, directed approach, carefully designed stroke placement steps
- can create high-quality results quickly
- not usually defined in terms of energy function
- general approach:
  - repeatedly placing strokes without ever changing them
  - iteration guided by where the next stroke will be placed
  - iteration guided by what shape the next will have
- a number of approaches discussed based on answers to these questions

Single-Point Strokes

- sampling the image only at one point, color assigned to entire new stroke
- parameterization of stroke shapes (lines, dots, many other shapes)
- parameterization of stroke sizes
- interactive placement of strokes (human control)
- also automatic stroke placement possible
- try it out: http://laminadesign.com/explore/impression/

Single-Point Strokes



Single-Layer Painterly Rendering

- generalization of single-point stroke technique
- input: source image and orientation field
- strokes are placed based on a grid that is overlaid
  - specific stroke positions slightly perturbed
  - strokes oriented depending on the orientation field
  - strokes drawn in random order to remove regularities

Single-Layer Painterly Rendering

Multiple-Layer Painterly Rendering

- also a form of single-point strokes (1 sample point)
- based on observations on painting approaches
  - start with a rough sketch, background
  - successively add detail with finer brushes where needed
- algorithmic realization
  - image constructed of layers with increasing detail
  - increasing detail: brush size divided by 2 each time
  - reference image for each layer created by matching blur
  - paint only where reference image differs from source
  - thus, detail is only added where necessary

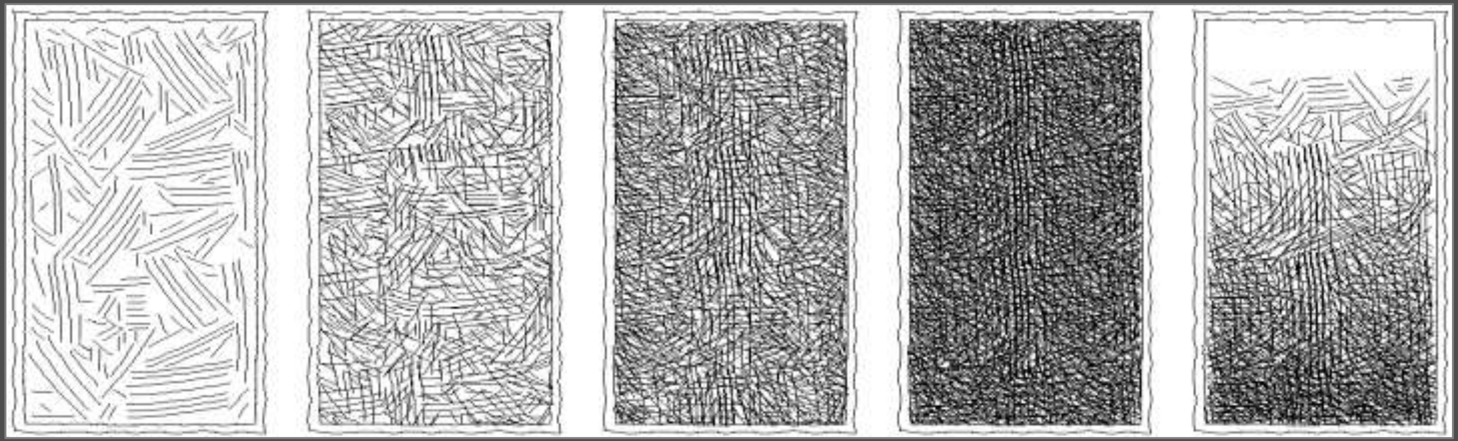
Multiple-Layer Painterly Rendering

Painterly Rendering: Long, Curved Strokes

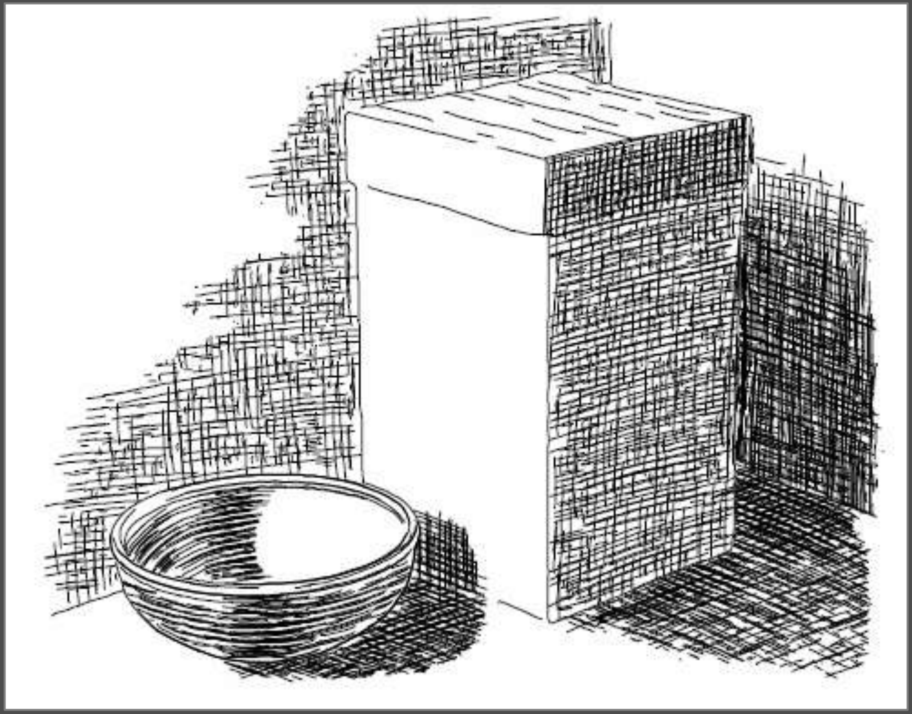
- as in the example, use longer strokes as in real paintings
- stroke samples color at one spot in the image
- trace strokes along a certain chosen direction, for example:
  - gradient of the image luminance
  - edge extraction and direction field interpolation (mosaics)
- quit after maximum length or color difference large
- stroke model to more closely model reality

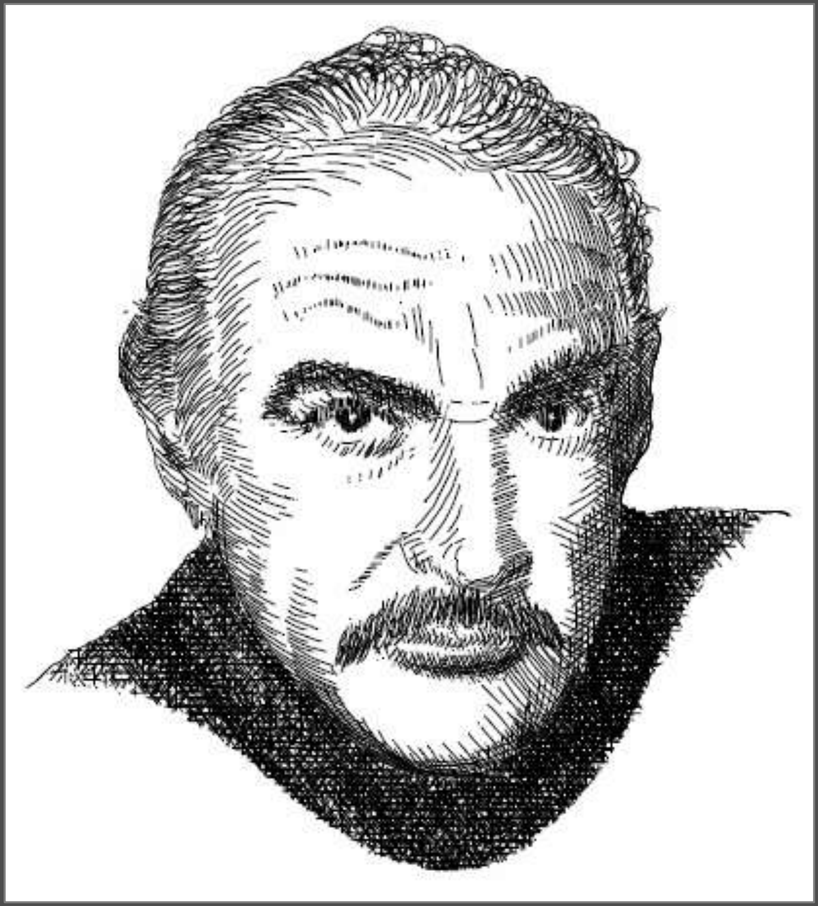
Pen-and-Ink Techniques as SBR

- prioritized stroke textures and extensions
  - match gray level tone of target image
  - achieve a target texture
  - achieve a target stoke orientation
- stroke placement guided by pre-defined sets
  - strokes selected in given order
  - possibly re-oriented to match target properties



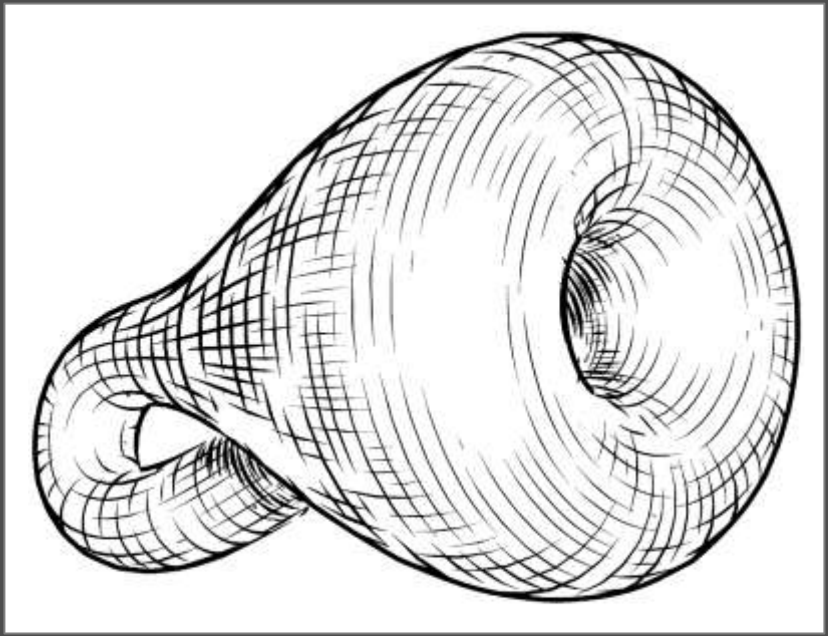
Pen-and-Ink Techniques as SBR

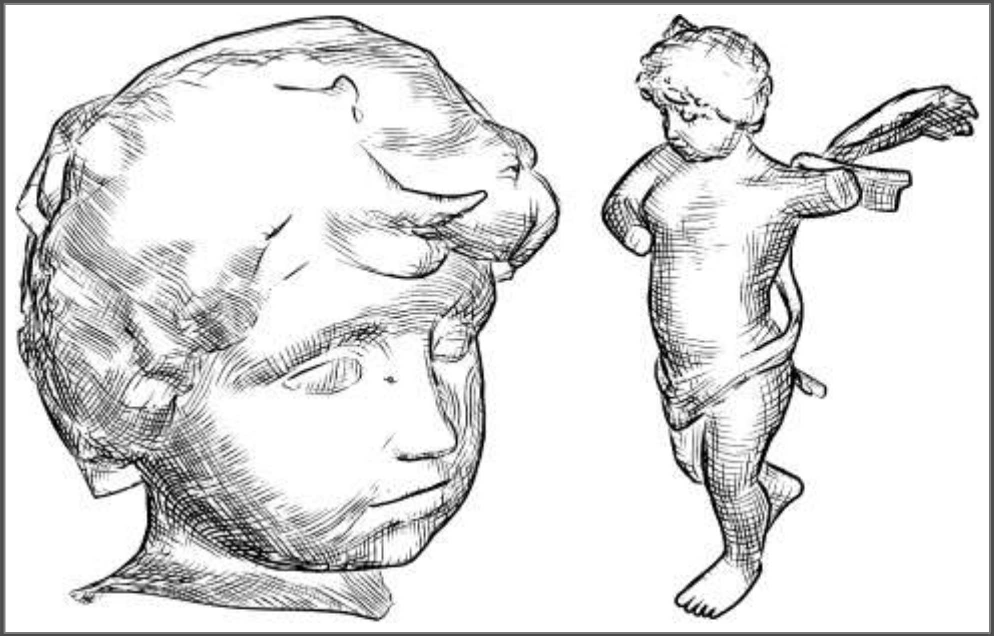




Pen-and-Ink Techniques as SBR

- hatching of 2D images similar
  - stroke placement guided by direction field extracted from curvature on 3D shape
  - strokes traced along these directions, evenly spaced
  - stroke parameterization to match target tone



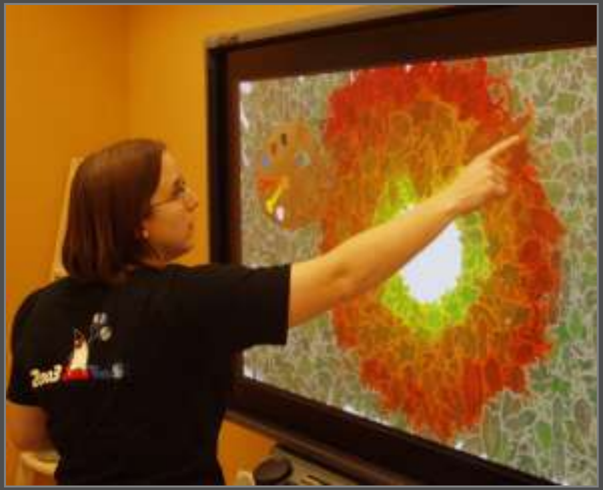


Stroke-Based Rendering

Summary

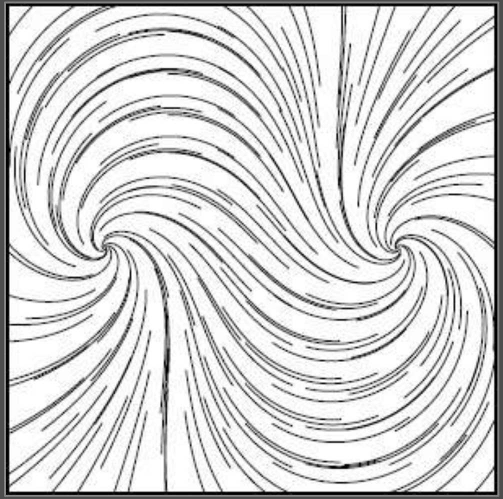
Stroke-Based Rendering: Summary

- SBR: stroke as a primitive for NPR
- use of energy function to describe goal to achieve
- not all target styles can be formulated this way
  - e.g., difficult to capture looseness or sketchiness
  - also, artistic target styles are not deterministic
- thus, direct (greedy) approach sometimes easier
- automatic techniques not always best
  - (intuitive) artistic control good and necessary
  - more on this later

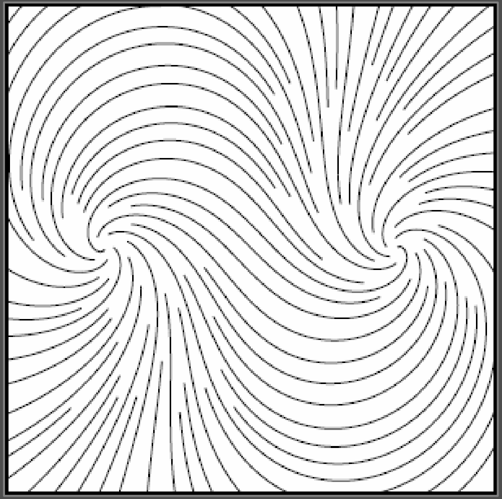


SBR: Applications Beyond NPR

- applications wherever strokes are used
- example: streamline visualization for vector fields
  - goal: even placement of streamlines in a vector field

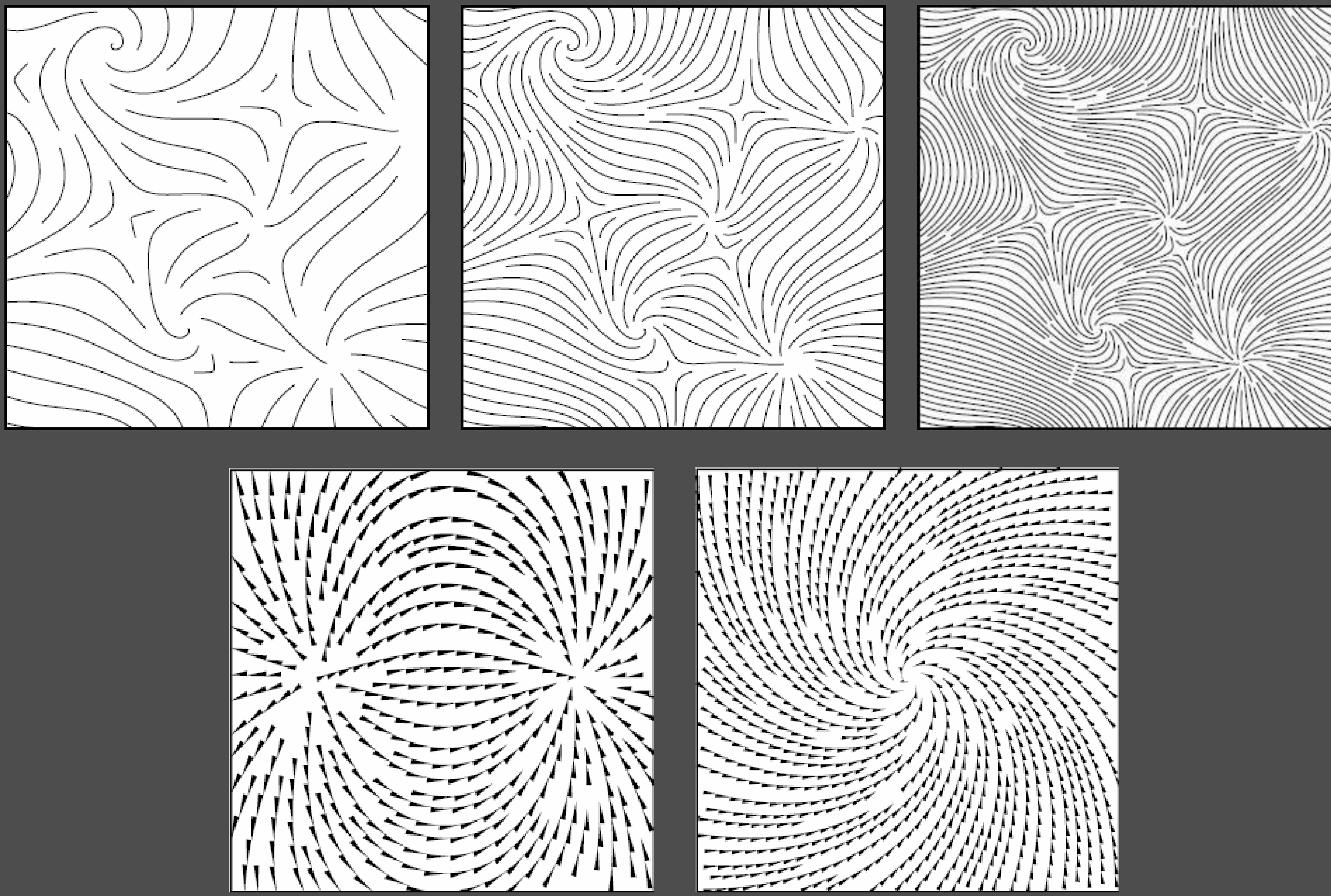


→



  - greedy approach, similar to 2D hatching technique
  - challenges: even distribution over entire image while creating long lines that are not interrupted

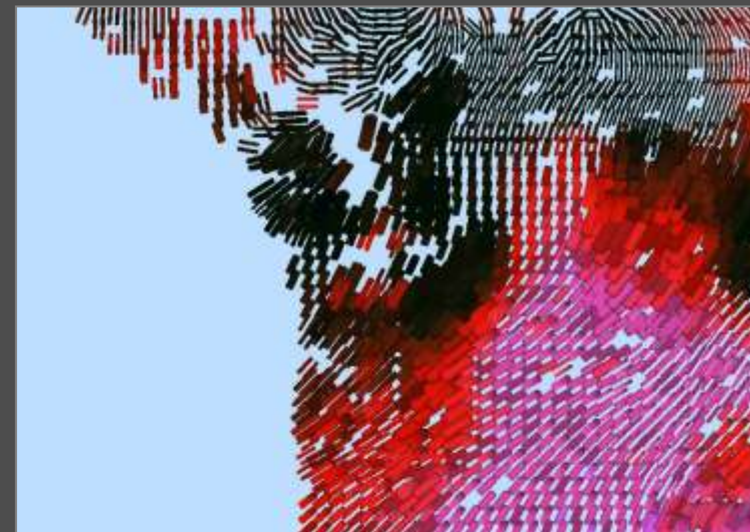
SBR: Applications Beyond NPR



SBR: Applications Beyond NPR

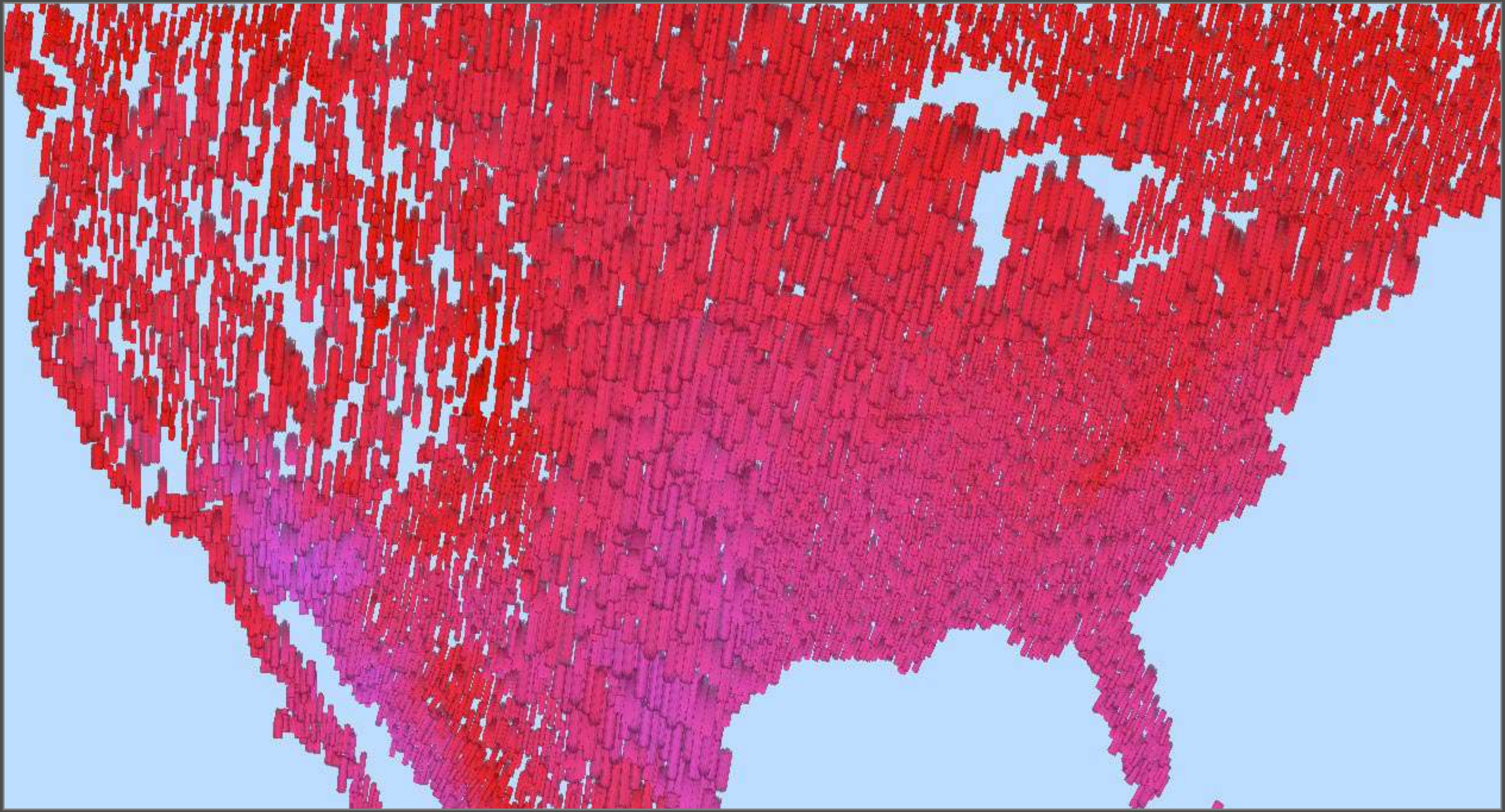
- vector and data visualization as painterly rendering

→



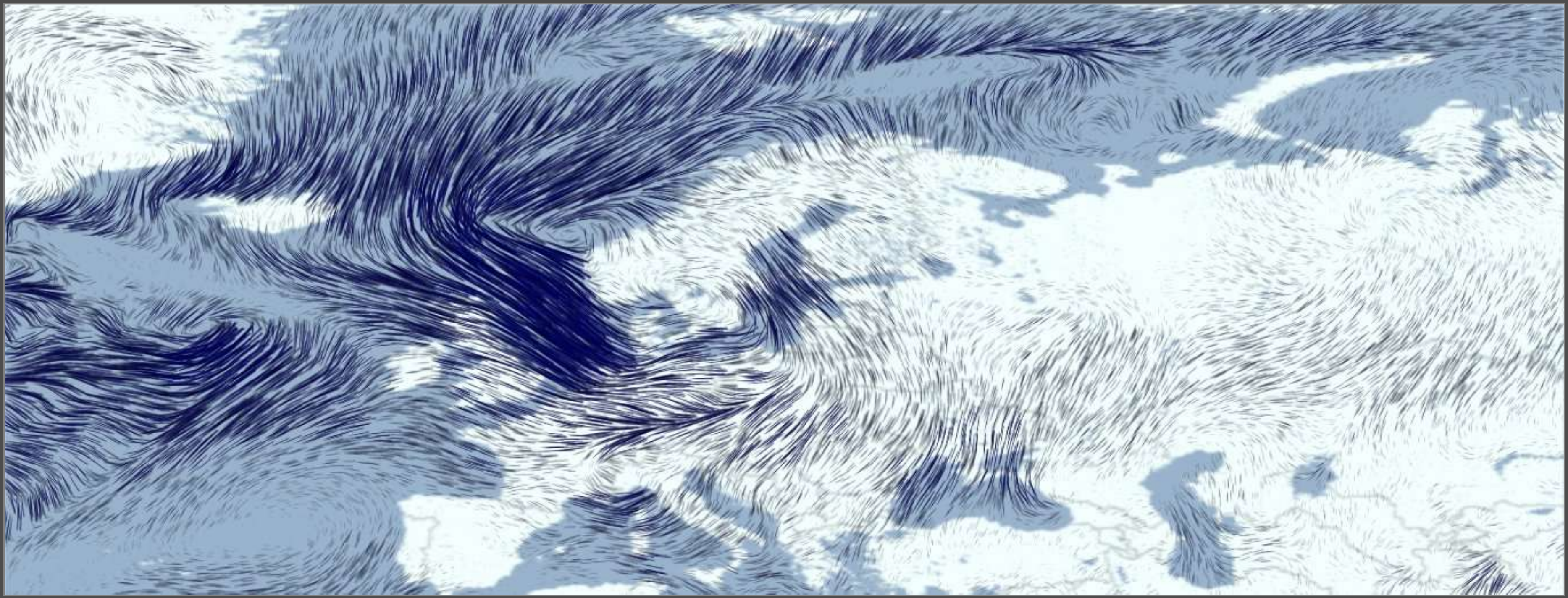
  - energy minimization trial-and-error approach
  - based on minimizing overlap
  - shows temperature (color), wind direction (directionality), wind speed (coverage), pressure (size), precipitation (orientation)

SBR: Applications Beyond NPR



SBR: Applications Beyond NPR

- interactive stroke-based vector visualization
  - direct interactive placement
  - sampling wind speed (size) and direction (orientation)



Approaches Related to SBR

- many traditional techniques are based on strokes
  - watercolor painting, Asian painting styles
  - oil painting, etc.
- stroke placement not as important
- focus on simulation of natural media/materials