Dense Image Over-segmentation on a GPU

Alex Rodionov

4/24/2009

Outline

1) Problem/Motivation

2) Goals

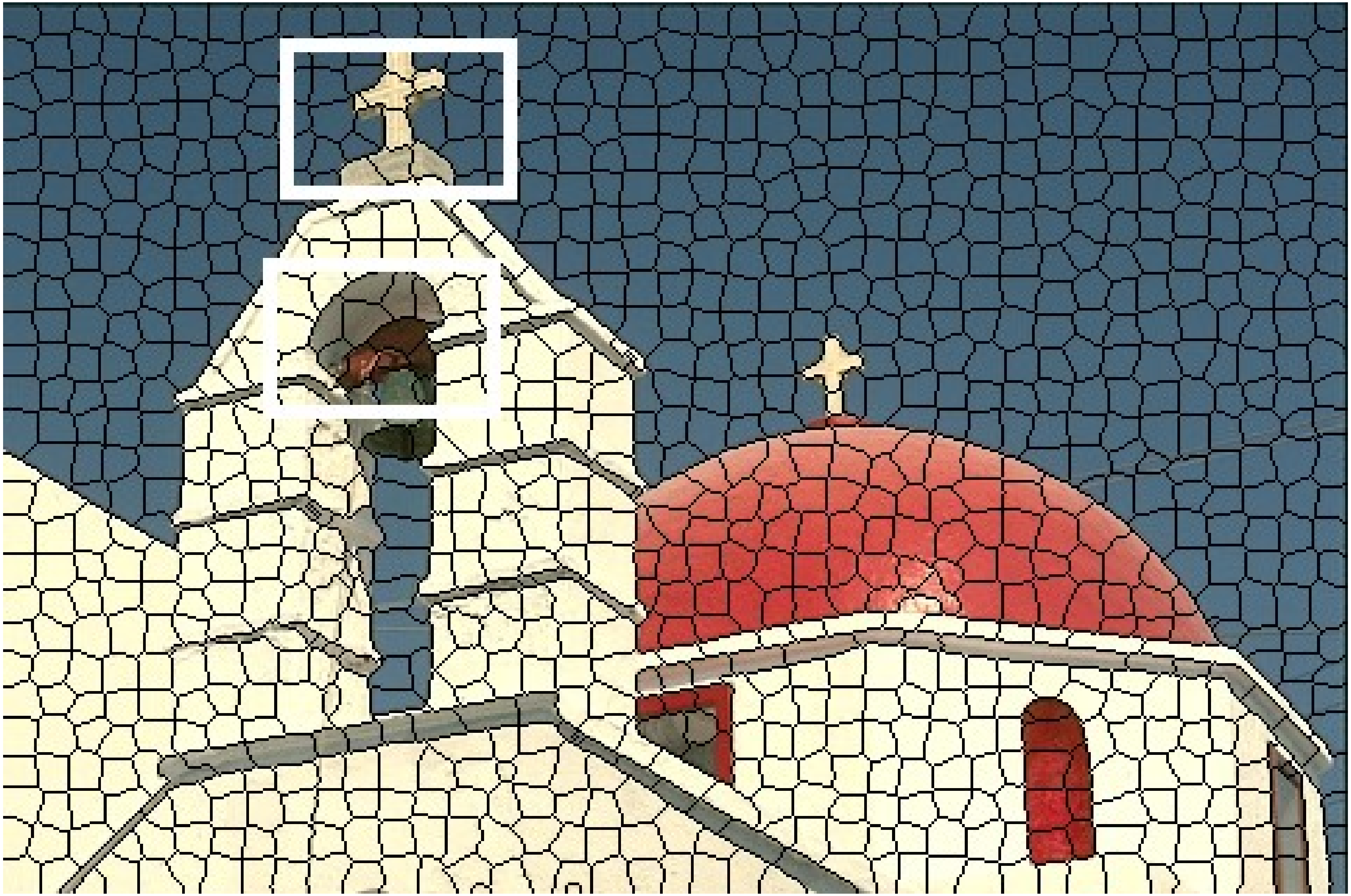
3) Overview of Algorithm

4) Implementation

5) Results

6) Future work & Conclusions

Problem



Applications

- Computer vision (object recognition, intelligent segmentation, ...)
    - Apply Algorithm X to a grid of superpixels vs. a grid of pixels
- Image compression

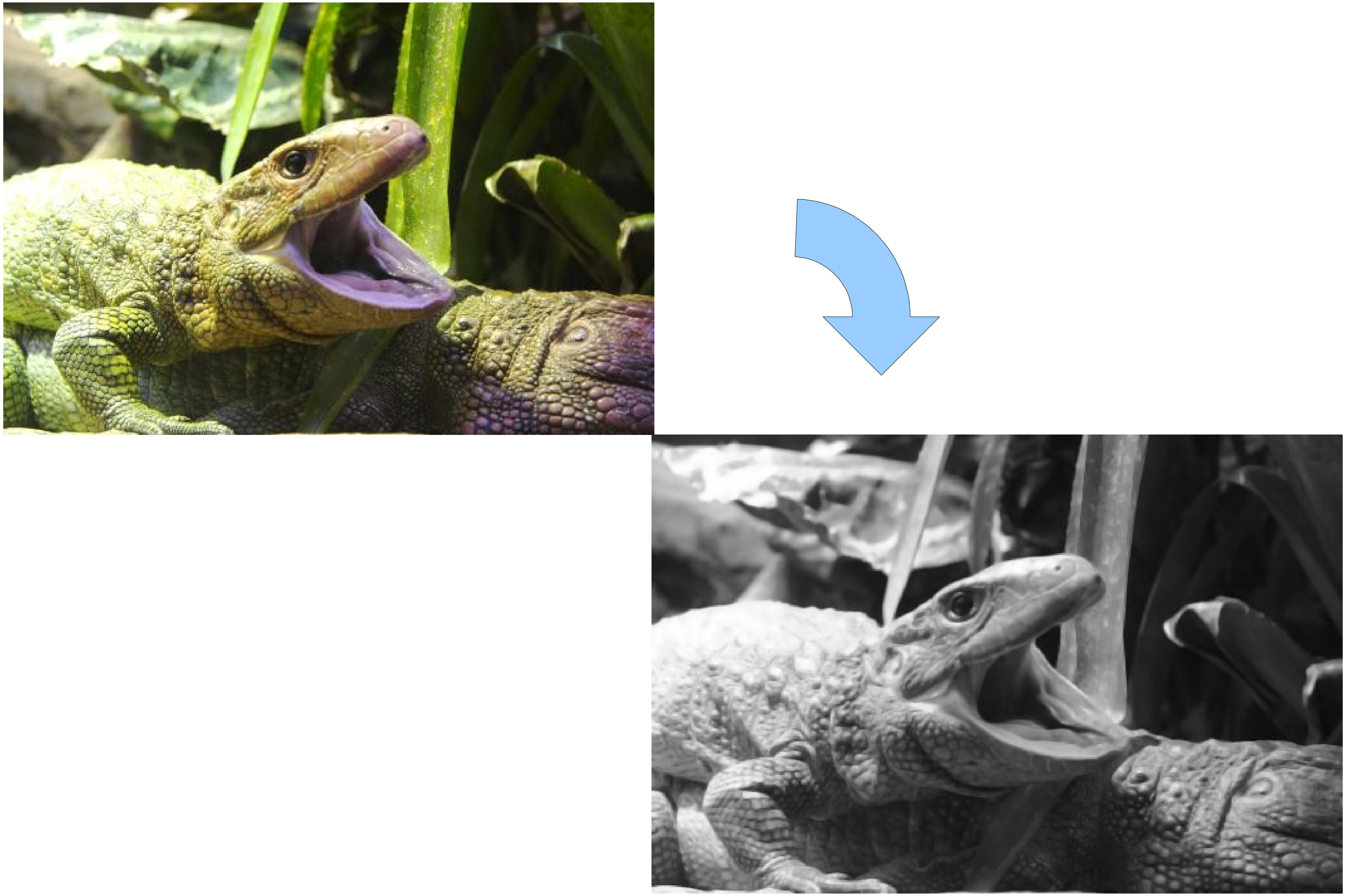
Goals

- Run at interactive framerates
- Fast enough for interactive use (video)
- Segment a 640x480 image in 200ms or less (5+ FPS)

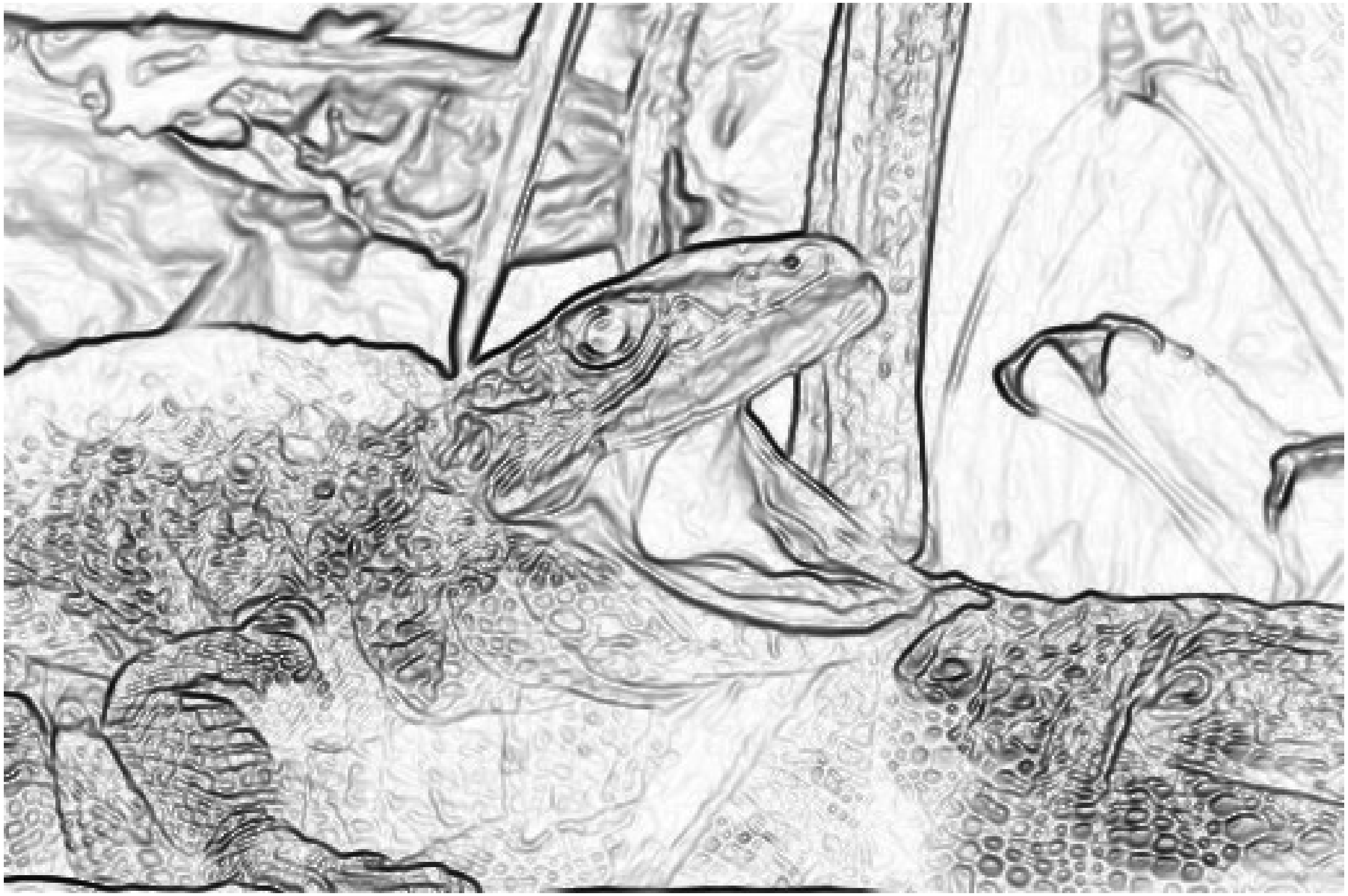
The Algorithm

- 0) Preprocess image (to grayscale, smooth)
- 1) Calculate speed map
- 2) Place N seed points throughout the image
- 3) Initialize a distance function to these seeds
- 4) Evolve distance function one timestep
- 5) Superpixel boundaries: pixels where distance func == 0

0) Preprocess

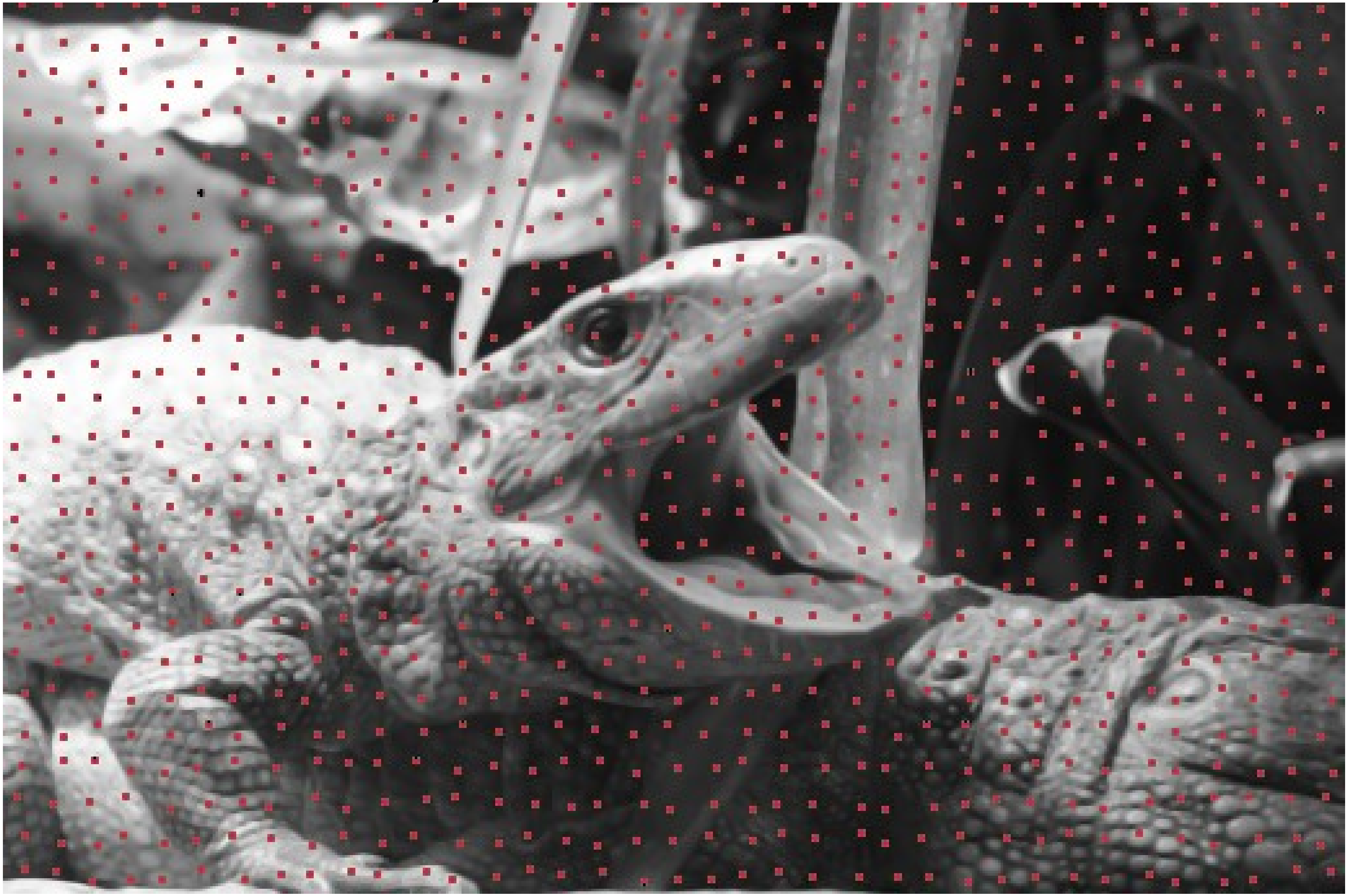


1) Calculate Speed

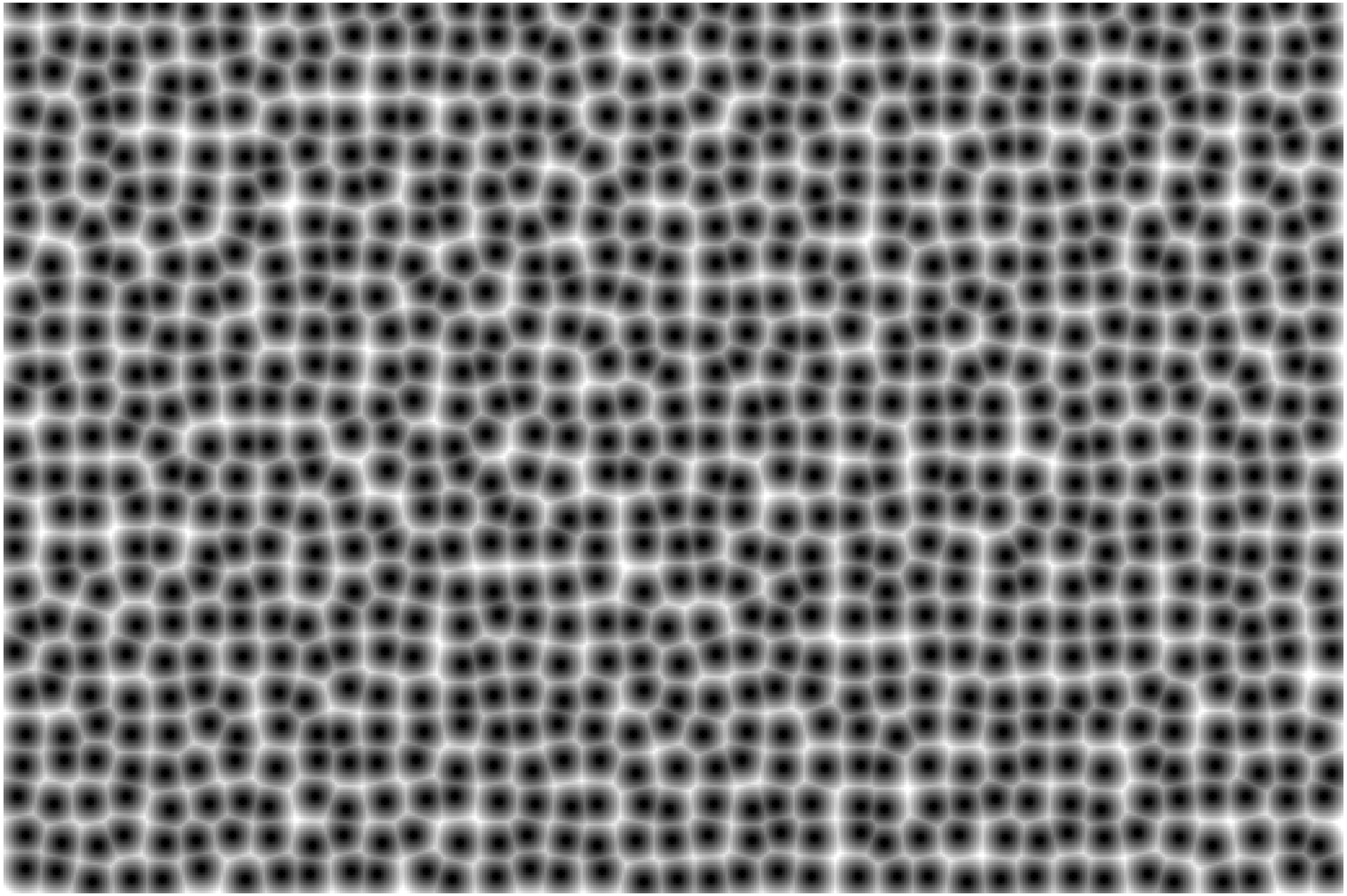


- Function of image gradient magnitude (edge strength)

2) Place seeds

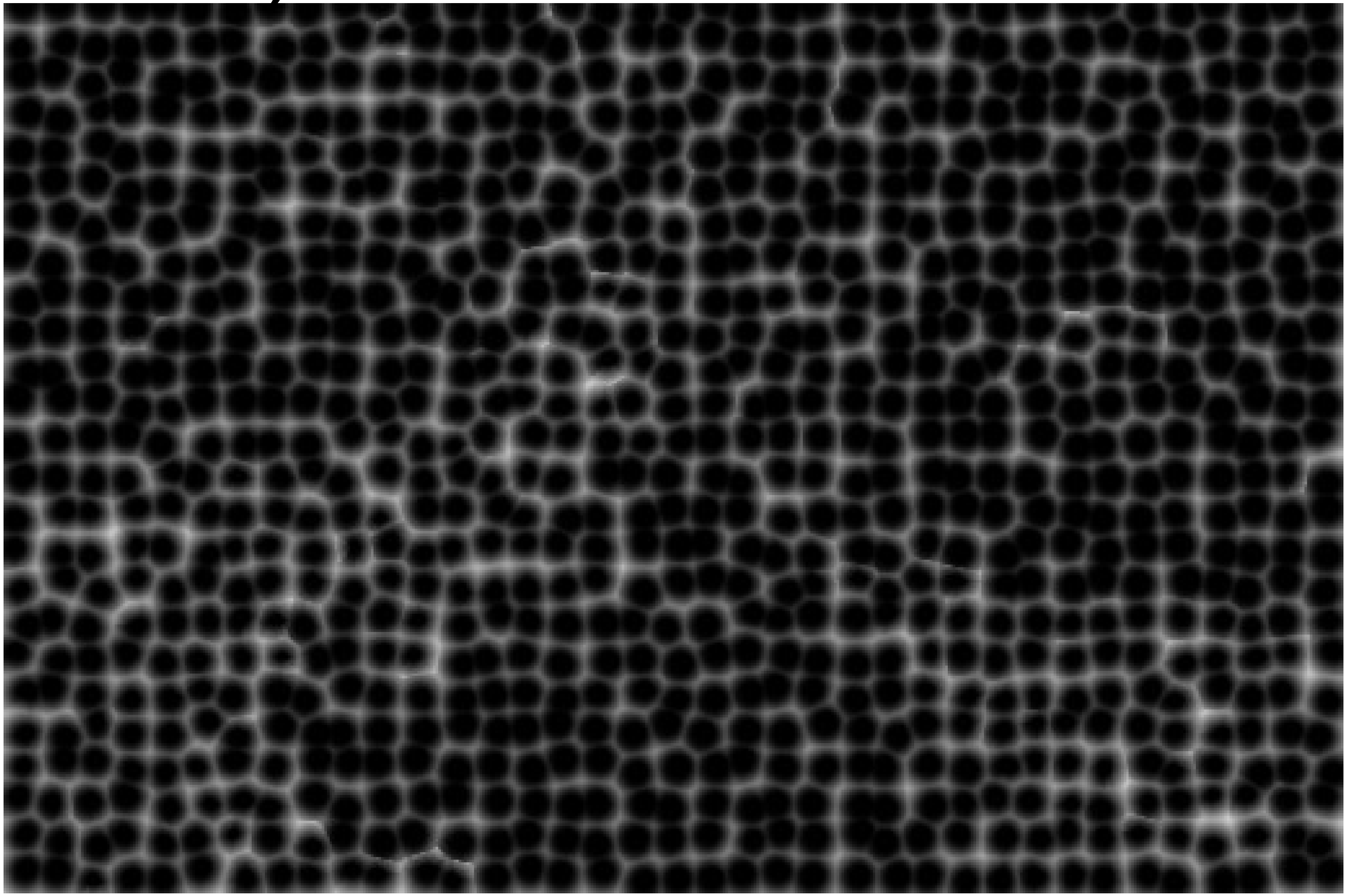


3) Init Distance Function



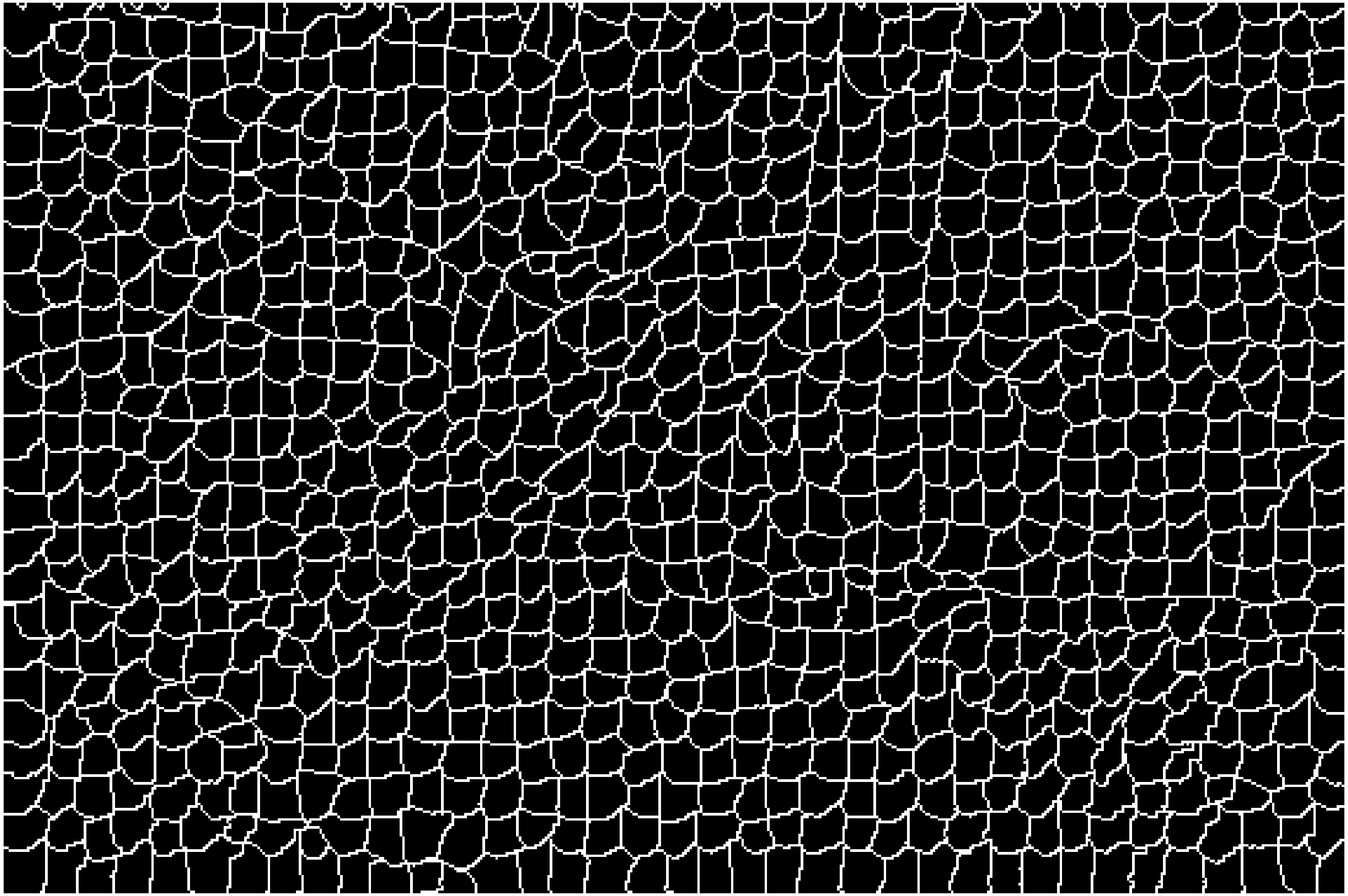
Pixel value = distance to closest seed

4) Evolve the function



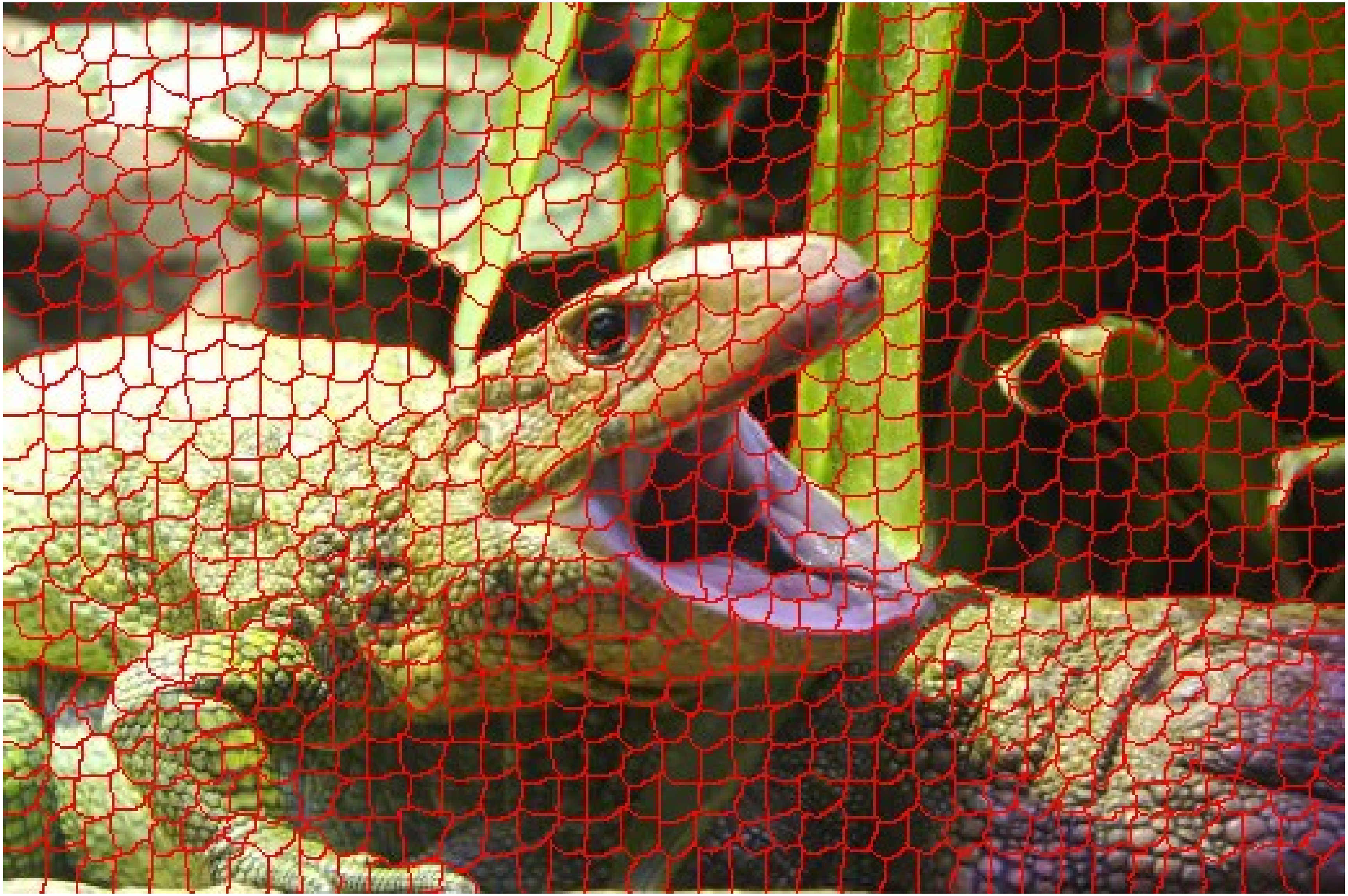
- Superpixels grow over time
- Evolution done by partial differential equation
- d(pixel)/dt function of spatial derivatives, speed, and proximity to other superpixels

5) Extract boundary



Zero-crossings of distance function define boundary

Result



GPU Implementation

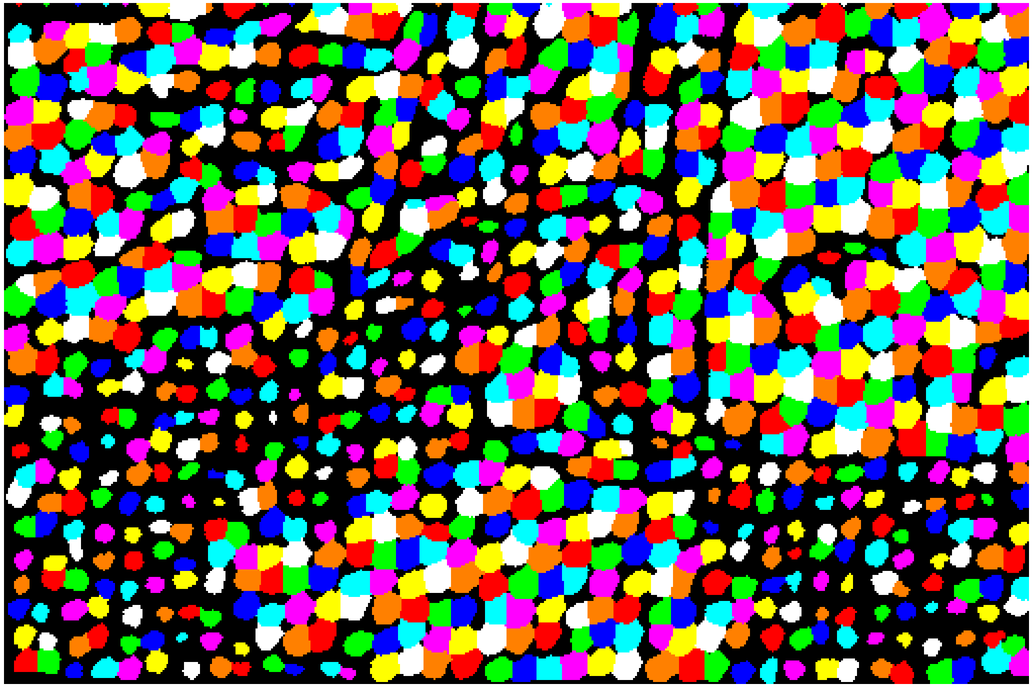
- Original implementation of TurboPixels done in MATLAB, parts accelerated with C
- GPU Implementation is C++ accelerated with CUDA
- Not all parts of original algorithm mappable to GPU (algorithms not parallel!)

GPU Implementation

- Example: Distance Transform
  - Foreach pixel, get distance to nearest pixel-of-importance
  - Used in initializing distance function
  - Used during evolution to get nearest superpixel boundary point
- Original algorithm used Fast Marching Method to calculate (uses global data structure, not parallel)
- Replace with a GPU-friendly substitute

GPU Implementation

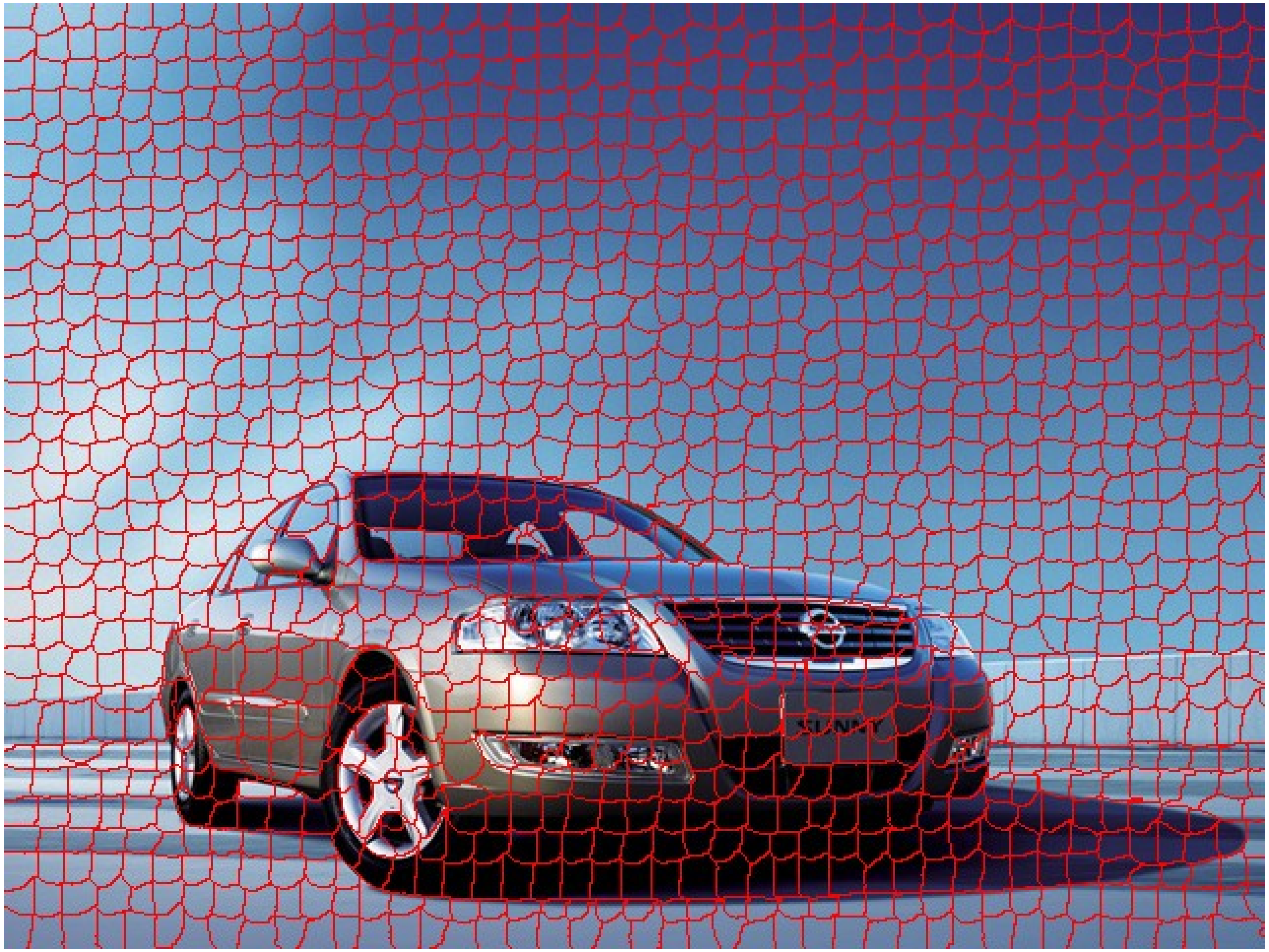
- Assignment map
  - Array of superpixel IDs
  - Keeps track of superpixel coverage, ownership
  - Prevents merging of superpixels



Performance Optimizations

- Use CUDA arrays,textures when kernel performs random or neighbor accesses
- Little use of shared memory
    - Kernels either read once+write once, or have complex access patterns not easily done with shmem
- Loop unrolling, aligning image array sizes, ...

Results



Sample Result

- Platform: NVIDIA GTX280
- Image size: 640x480
- Time: 443ms (2.25FPS)
- Timesteps: 122
- Superpixels: 1000

- Software implementation: 30 sec on 481x321 image

Conclusions

- Implemented a TurboPixels-like image oversegmentation algorithm on a GPU
- Performance goal of 5fps on 640x480 not quite attained
- Achieved significant speedup over software implementation (although one written mostly in MATLAB...)

Future Work

- Algorithmic optimizations:
  - Evolve area around the expanding boundary, instead of evolving everything
  - Use variable-length timesteps to reduce number of timesteps and amount of work
- Application to video:
  - Once it runs fast enough, then what?
  - Modify algorithm to take advantage of inter-frame coherence