GPU-ACCELERATED SNAKE

GPU IMPLEMENTATION OF A REGION-BASED SEGMENTATION ALGORITHM (SNAKE) FOR LARGE IMAGES

Gilles Perrot[1], Stéphane Domas[1], Raphaël Couturier[1], Nicolas Bertaux[2].

[1]University of Franche-Comté - Distributed Numerical Algorithmics group (AND),
[2]Fresnel Institute - Physics and Image Computing group (PhyTI)

31 august - 1 september

Table of contents

Image segmentation

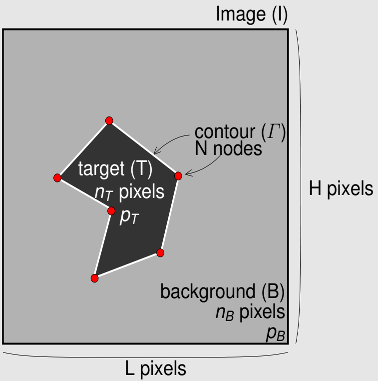
Definition, goal

- Dividing an image into two homogeneous regions.
- Reducing the amount of data needed to code information.
- Helping the human perception in certain cases.

Image characteristics

- 16 bit-coded gray levels,
- From 10 Mpixels to more than 100 Mpixels,
- Very noisy.

Algorithm basics : criterion

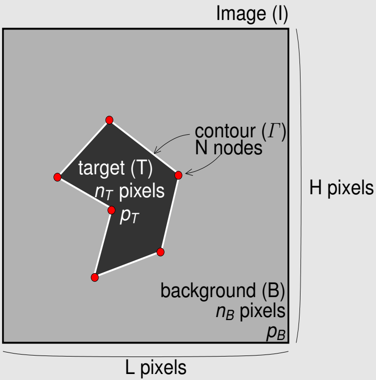


- The goal is to find the most likely contour Γ (number and positions of nodes).
- The criterion used is a *Generalized Likelihood*.
  In the Gaussian case, it is given by

$$GL = \frac{1}{2}\left[n_B.log\left(\widehat{\sigma_B}^2\right) + n_T.log\left(\widehat{\sigma_T}^2\right)\right]$$

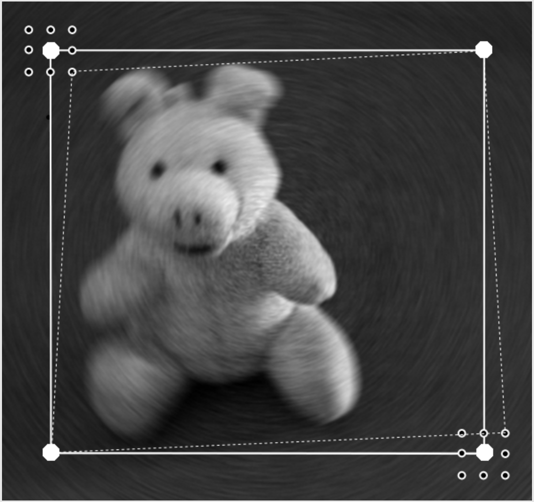
where $\widehat{\sigma_\Omega}$ is the estimation of the deviation σ for the region Ω.

Algorithm basics : criterion



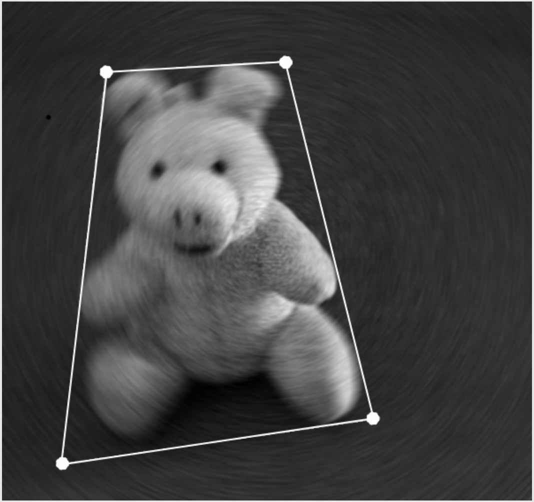
- Parameters estimation is done by 1-D sums on along the contour.
- Every pixel coordinates are needed.

Snake algorithm in action



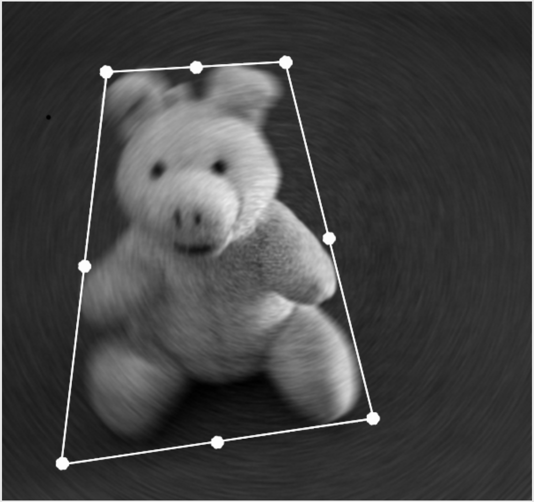
- 150 Mpixels image.
- Initial contour: 4 nodes.

Snake algorithm in action



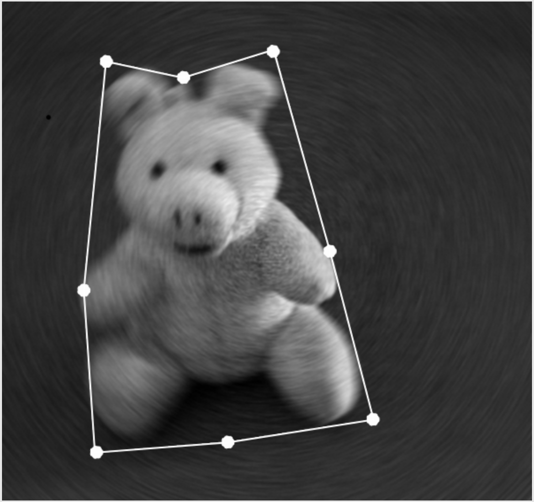
- End of first iteration: no more move can be of interest.

Snake algorithm in action



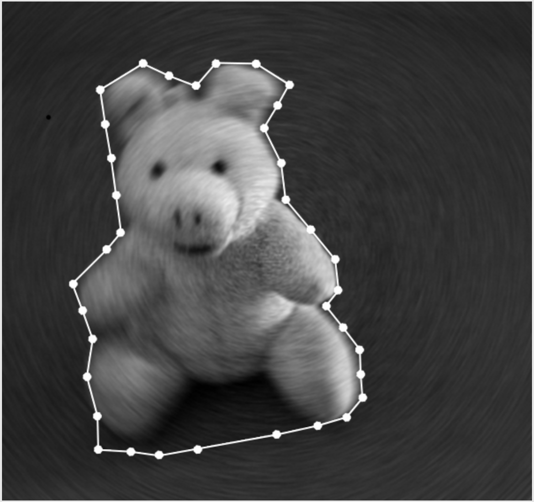
- Nodes added in the middle of segments.

Snake algorithm in action



- End of second iteration.

Snake algorithm in action



- End of fourth iteration

Snake algorithm in action

- End of seventh iteration.
- Fast segmentation.
- Efficient with noise.

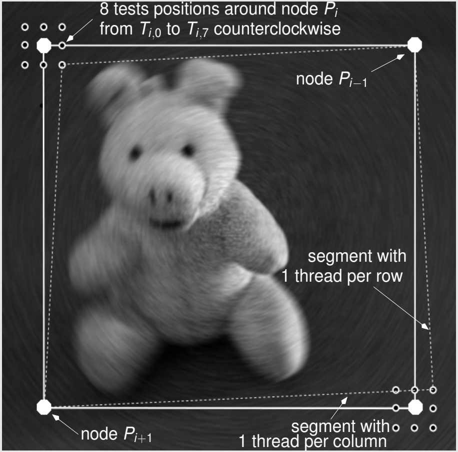
GPU : why

- This SNAKE algorithm has proved to be fast and robust.
- Images to be processed are becoming larger and larger.
- To be user-friendly, process must be done in less than 1 second.
- GPU are cheap and can bring impressive speedups.
- GPU are easy to embed in a simple PC (in a aeroplane,...)

GPU design : key points

- The parallelism of a modern GPU lays on a SIMT paradigm (Single Instruction Multiple Threads): the same instruction is processed by a great number of threads at a time (up to 2^{16}).
- Threads are compounded in independants blocks with no possible synchronization between blocks.
- Threads in a block share a small amount of shared memory (16-48 KBytes).
- There are restrictive conditions to be fullfilled in order to make efficent accesses to global and shared memory.
- Data transfers between CPU and GPU are slow.

GPU implementation: parallelization



- Every 16 segments for every node are processed in parallel.
- Fits GPU specific parallelism: each segment pixel is processed by a thread.
- Criterion values are obtained after several reduction stages.
- All nodes are possibly moved in one step.

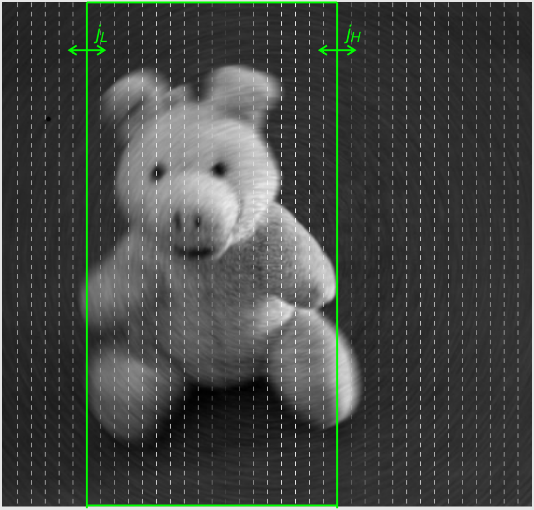
GPU implementation: first results

- Global speedup around x7-x8 (Nvidia C2050) for image sizes from 15 to 150 Mpixels.
- First iterations have higher speedups:
  - several large segments,
  - few inactive threads in the grid.

GPU implementation: smart init (reasons)

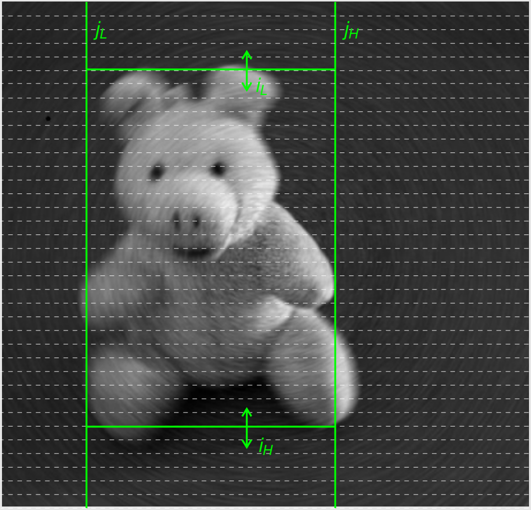
- The target shape is often far from initial contour,
- It causes the very first iteration to be much more time-consuming than the other ones.
- It's fast on GPU to find a rectangle near the target. But it needs to overrule the 'one thread/one pixel' principle.

GPU implementation: smart init (process)



- Realize a periodic sampling of a few hundreds of J-coordinates.
- Evaluate in parallel every possible rectangle of diagonal $(0, j_L) - (H, j_H)$.
- Select the one with the best GL criterion.
- j_L and j_H are now considered as constants.

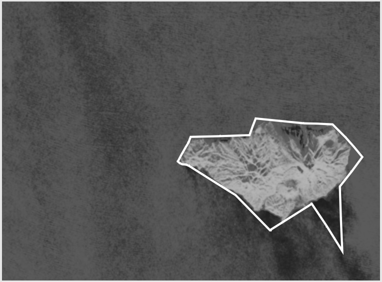
GPU implementation: smart init (process)



- Given j_L and j_H.
- Realize a periodic sampling of a few hundreds of I-coordinates.
- Evaluate in parallel every possible rectangle of diagonal $(i_L, j_L) - (i_H, j_H)$.
- Select the one with the best GL criterion.

GPU implementation: improvement

- Global speedup around x10 (Nvidia C2050) for image sizes from 15 to 150 Mpixels and a 'small enough' target.
- Less than 0.6 s for the 150 Mpixels image of the example.
- A real-life 10 Mpix S.A.R. picture after 3 iterations in less than 50 ms .



Conclusion, future works

- Interesting speedups
- Original algorithm is not GPU-friendly
- Future works:
  - Finding a more suited structure to describe the contour.
  - Switching to a statistical model independant from a PDF: the potts model.
  - Benefit from recent features of CUDA v4 (overlapping, multiple kernels).
  - Extend to a multiple targets algorithm, based on this single target elementary piece of code.