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

Images of our interest

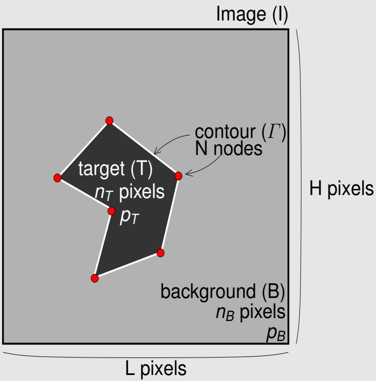
Origins

- Synthetic Aperture RADAR (S.A.R.),
- Ultrasonic (medical imaging),
- Photographic (IR, nightshots).

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* one .
  In the Gaussian case, it is given by

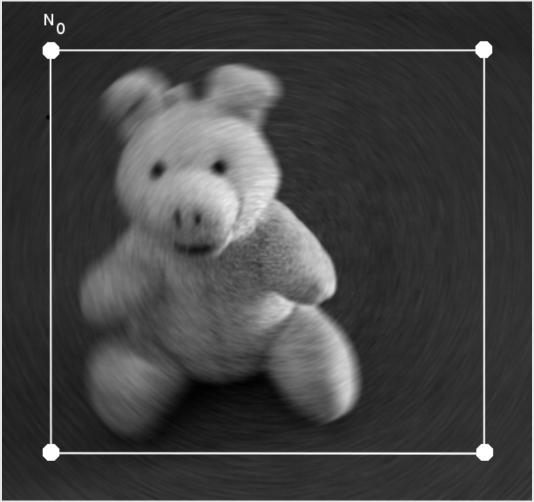
$$GL = \frac{1}{2}\left[n_B.log\left(\widehat{\sigma_B}^2\right) + n_T.log\left(\widehat{\sigma_T}^2\right)\right]$$

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

Algorithm basics : parameters estimation

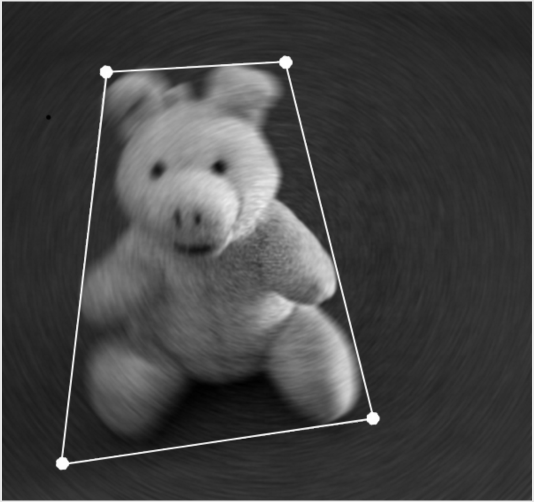
- Based on the Green-Ostogradsky theorem, Chesnaud has shown how to replace those 2-dimensions sums inside the contour by 1-dimension sums along the contour.
- This optimization implies:
  - the precomputation of a few matrices (called cumulated images) containing the potential *contributions* of each pixel of the image,
  - the use of constant lookup tables of weighting coefficients to determine the *contributions* of each segment of pixels.

Snake algorithm in action



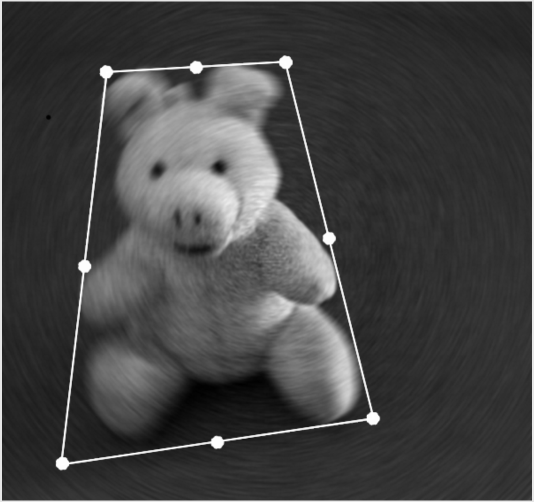
- 15 Mpixels image (SSE implementation limit).
- Initial contour: 4 nodes.

Snake algorithm in action



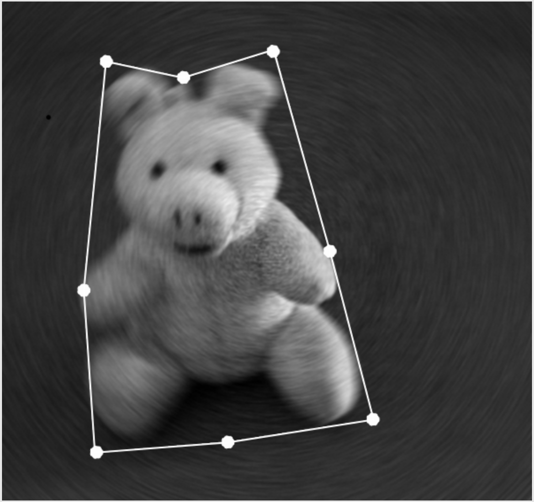
- End of first iteration: no more move can be of interest.

Snake algorithm in action



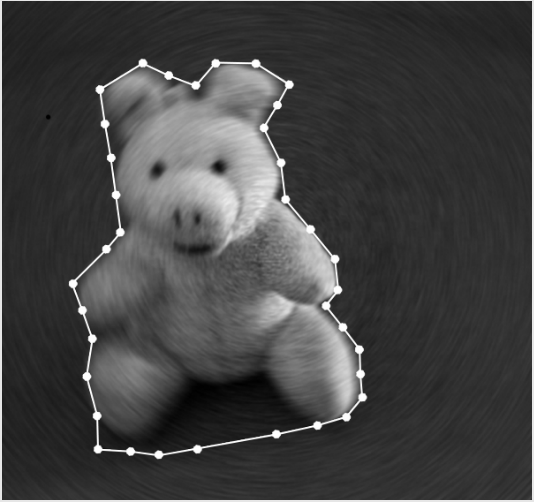
- Nodes added in the middle of segments.

Snake algorithm in action



- End of second iteration.

Snake algorithm in action



- End of fifth iteration (36 nodes).

GPU implementation: prior knowledge

- 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: precomputations

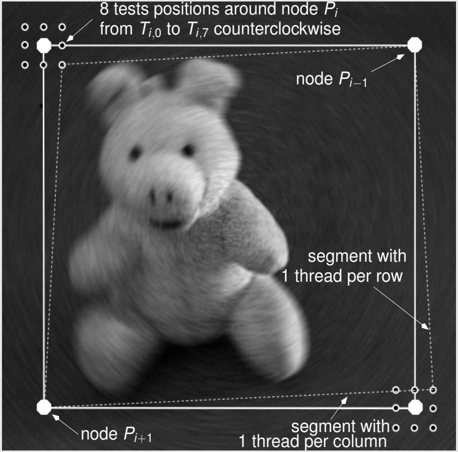
- One of the cumulated images is not to be computed anymore: values are evaluated on the fly.
- An inclusive parallel prefixsum is performed on each row of the image for each matrix to be processed (z,z^2).
- Speedup is around x7 for images larger than 100 MPixels. Comparison is done with the SSE/CPU implementation of the PhyTI group.
- Higher speedups (x15) are obtained with specific versions for constant image sizes.

GPU implementation: nodes move

To select the possible next position of a node:

- Parameters of the corresponding contour have to be estimated.
- Then the value of the criterion can be obtained and compared with the previous one.
- The parallelization needs reside essentially in the parameters estimation.
  Two possible parallelism levels:
  - One contour per thread.
  - One pixel per thread.
  - ➽ The *one pixel per thread* rule is far more efficient, due to memory access constraints.

GPU implementation: parallelization



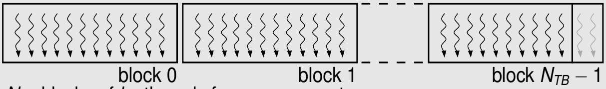
- Every 16 segments for every even/odd nodes are processed in parallel.
- Fits GPU specific parallelism: each pixel is processed by a thread.

GPU implementation: data structure

The main idea is to organize, in a single array, every pixels of every segments to be processed.
Thus, for a given state of the contour (N nodes), we:

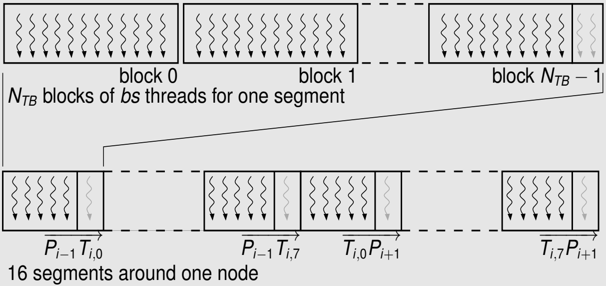
1. Find the largest segment to be processed. It gives:
   - the block size bs of the computing grid,
   - the number of blocks needed for each segment (N_{TB}).
2. Compute in parallel, the coordinates of every pixels of the $16.N$ segments to be considered,
3. Make some parallel reductions to finally obtain parameters estimation.

GPU implementation: data structure

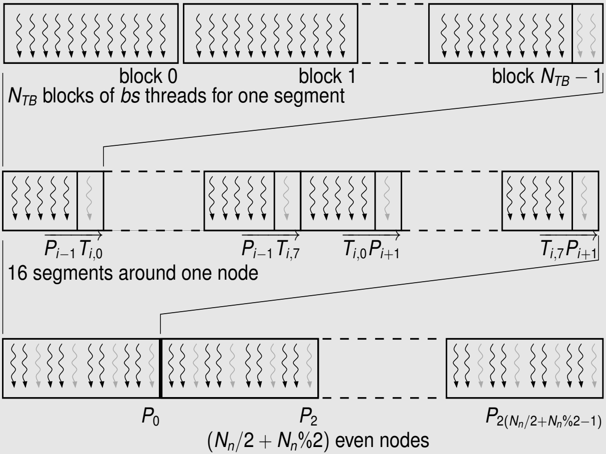


N_{TB} blocks of *bs* threads for one segment

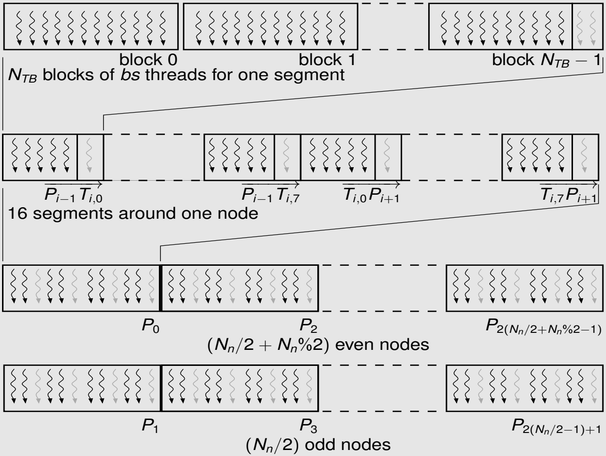
GPU implementation: data structure



GPU implementation: data structure



GPU implementation: data structure



GPU implementation: first results

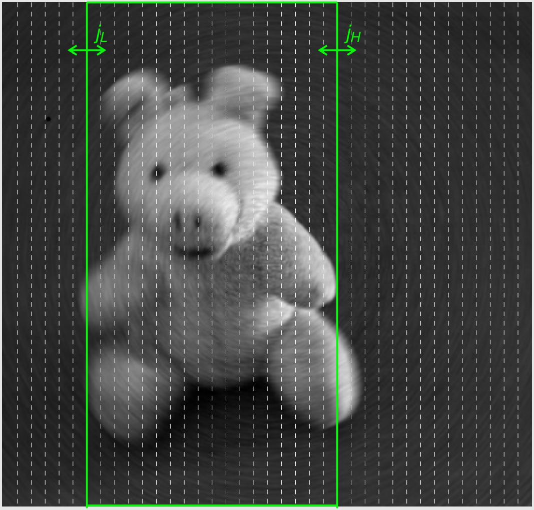
- Global speedup around x7-x8 for image sizes from 15 to 150 Mpixels.
- First iterations have higher speedups:
  - several large segments,
  - few inactive threads in the grid.
- Last iterations are sometimes slower than on CPU:
  - a lot of small segments,
  - more inactive threads in the grid.

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.
- Horizontal segments contributions are null.
- Vertical segments contributions computations can be fast, through a specific process.

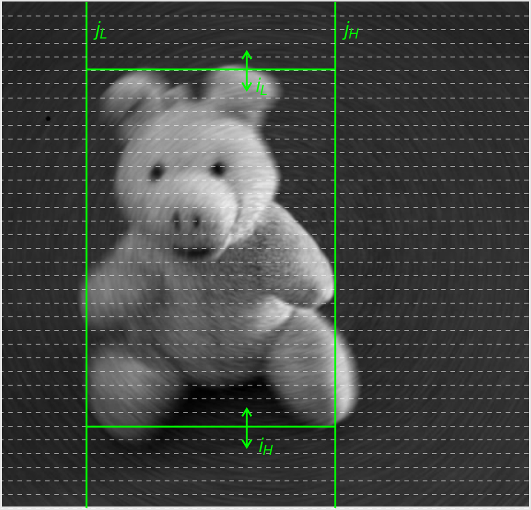
➠ It's fast to find a rectangle near the target.

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: enhancement

- Global speedup around x10 for image sizes from 15 to 150 Mpixels and a small enough target (as in the example)
- Less than 0.6 second for the 150 Mpixels image of this example.

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.