Tile-based Image Forces for Active Contours on GPU

Enrico Kienel and Guido Brunnett

Chemnitz University of Technology, Germany

Abstract

Active contours have been proven to be powerful semiautomatic image segmentation tools. We present an adaptive image force computation scheme in order to minimize both computational and memory requirements. Hence, we are able to perform a fast semiautomatic contour detection in huge images. We present an efficient implementation of this approach on the basis of general purpose GPU processing providing a visual continuous active contour deformation.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Image Processing and Computer Vision]: Image Processing and Computer Vision—Edge and Feature Detection

1. Introduction

Active contours are powerful semiautomatic image segmentation tools that are especially useful in the extraction of complex object boundaries [KWT87]. They have been proven to cope with different image modalities, arising from their superior characteristics, e.g. robustness against noise and tolerance against incomplete object boundaries.

In this paper, we focus on the adaption of fast parametric active contours for huge images. The paper is organized as follows. We give a brief overview of the active contour model in Section 2. Our contributions comprise a tiling approach in combination with a spatially constrained image force computation scheme on demand, that is explained in Section 3, and its efficient GPU-based implementation, described in Section 4. We demonstrate the efficiency of our method by giving comparative results in Section 5 and finally conclude this paper in Section 6.

2. Active contours

Active contours are often referred to as snakes, because of their typical behaviour to smoothly move across the spatial domain of an input image, driven by an iterative energy minimization. In the original publication snakes are modelled as parametric unit speed curves $\vec{v}(s) = (x(s), y(s))$ that are defined on the spatial domain of a given image I. A non-trivial energy functional has to be minimized in order to drive the snake towards salient features of interest, e.g. image edges. The total energy term $E(\vec{v}) = E_{\text{int}}(\vec{v}) + E_{\text{img}}(\vec{v}) + E_{\text{con}}(\vec{v})$, that is assigned to a snake, consists of the internal energy E_{int}, the image energy E_{img}, and optionally a constraint energy E_{con}. Since an explanation of the partial energies as well as of the minimization would go beyond the scope of this paper, please refer to [KWT87, KVB06] for further details.

3. Tile-based image forces on demand

Basically, the image forces need to be computed only once for the entire image as it does not change over time. Hence, the image force $\vec{F}_{\text{img}}(x,y)$ at the location of every control point of the snake $\vec{v}(s) = (x,y)$ at any time of the deformation can be determined by a simple lookup. Moreover, several contours within the same image can be extracted subsequently using the precomputed image forces. However, the increasing level of detail needed in today's applications leads to image sizes that are difficult to handle by off-the-shelf hardware.

Digitization of histological serial cross-sections in embryological research may produce huge images involving sizes of several hundred megapixels [SWKK02]. Therefore, the image force computation might easily reach the limits of available main memory of a standard desktop PC or at least, would take an unreasonable amount of time. However, the evaluation of image forces over the entire domain of an image is generally not mandatory, since the moving snake usually encounters only a small subset of the image.

Thus, we propose a simple tile-based strategy. According to an overlaid uniform grid, we decompose the input image into $p \times q$ tiles, each having $n \times n$ pixels. The key idea is to dynamically compute the image forces of only those tiles, that an active contour currently explores. To this end, we modify the snake algorithm as follows:

- As soon as the snake enters a new grid cell, the image forces for this cell are computed. For this purpose, the continuous deformation, i.e. the iterative energy minimization, is temporarily paused.
- When a snake leaves a grid cell completely, the memory for the image forces will be released.

Considering almost convex contours, the memory as well as the complexity of the image force computation could be estimated with $\mathcal{O}(mn^2)$ rather than $\mathcal{O}(m^2n^2)$ for the entire image, with $m = \max\{p,q\}$. Hence, the predominant memory requirements for the storage of image forces are considerably reduced. Furthermore, we gain a significant speedup for the precomputation, as the image forces need to be computed at once only for the tiles that the snake initially covers. Note, that an image force update during the algorithm normally requires the computation just for single tiles.

On the other hand, this approach requires quite a lot of bookkeeping. The contour deformation has to be interrupted, whenever image forces are missing. (We take advantage of graphics hardware to compensate involved occasional delay effects, which is later described in detail.) Moreover, we have to keep track of the snake position relative to the elements of the grid. To efficiently manage the tile-based image energy, each grid cell basically stores the pixel-based distance `dist` to the closest snake point and a bit mask containing the following status flags:

- one `have`-bit indicating whether image forces *have been* already computed for the current cell
- one `need`-bit indicating whether image forces *need to be* computed for the current cell
- one `list`-bit indicating the presence of the current cell index in a sorted list L, that is maintained to provide quick access to cells that either *need* or *have* image forces

At the very beginning of the energy minimization, the distances of the grid cells are initialized with ∞ and $L = \emptyset$. The state of the cells must be updated before every deformation step as depicted in Algorithm 1.

Of course, one may argue that the algorithm seems to be slowed down, because the image forces are computed on demand *during* rather than *before* the energy minimization procedure. Nevertheless, the overall performance gets basically improved as a result of only few grid cells that require an image force computation.

4. GPU assistance

The tile size $n \times n$ as well as the kind of implementation have a great impact on the performance of the image force computation. On the one hand, choosing n too big would lead to memory wasting and an enormous effort for the initial computation as well as for image force updates. On the other hand, choosing n too small leads to permanent reallocation and recomputation of image forces.

Algorithm 1: Pseudocode for efficient update of grid cell status

```
Input : Sorted list $L$ of relevant grid cells
1:  foreach grid cell $C \in L$ do
2:      $C$.need = false
3:      $C$.dist = $\infty$
4:  end
5:  foreach snake control point $\vec{v}_i$ do
6:      Identify grid cell $C$ containing $\vec{v}_i$
7:      $C$.dist = 0
8:      $C$.need = true
9:      if not $C$.list then
10:         Insert $C$ into $L$ using binary search
11:         $C$.list = true
12:     end
13:     foreach $C' \in N_8(C)$ do Update $C'$.dist with respect to $\vec{v}_i$
14: end
15: foreach grid cell $C \in L$ do
16:     if $C$.need then
17:         if not $C$.have then
18:             Compute $\vec{F}_{img}$ in $C$
19:             $C$.have = true
20:         end
21:     else
22:         if $C$.have then
23:             Delete $\vec{F}_{img}$ in $C$ and release memory
24:             $C$.have = false
25:         end
26:         Remove $C$ from $L$
27:     end
28: end
```

For a texture-based implementation on graphics hardware for both visualization and image force computation we set $n = 256$, which offers a reasonable trade-off between memory consumption and computation overhead as well as fast power-of-two texture processing. All $p \times q$ tiles are independently stored as mipmapped textures on graphics memory. Hence, no main memory is used for visualization purposes. Incomplete border tiles are padded with background pixels.

Huge images of histological cross-sections often suffer from visual artifacts und thus are strongly recommended to be preprocessed by different image filters to improve semiautomatic contour detection [KVK*07]. In our approach, both basic image processing and subsequent image force related computations are efficiently realized using programmable shaders on the GPU. As a nice side effect, our hardware-oriented implementation benefits from the tiling approach as it is not exposed to texture size constraints of underlying graphics hardware, and thus provides realtime visual interaction as well as fast pinpoint navigation capabilities. The image force field is finally read back into main memory to serve as look-up table for continuing snake deformation that is efficiently implemented in software [KVB06].

We propose Algorithm 2 to compute image forces from original image tiles stored as grayscale textures on graphics memory, which is illustrated in Figure 1. We perform these

GPU computations offscreen as there is no need for visualization. To this end, we simply render a required tile into a *frame buffer object* (FBO_a). We take advantage of SIMD capabilities of the GPU, i.e. to perform RGBA operations per fragment simultaneously for each channel. Thus, we combine four subsequent intensity values to one RGBA quadruple. With the aid of a *pixel buffer object* (PBO) the *internal-format* of the FBO_a content can be reinterpreted without an expensive bus transfer.

Another *frame buffer object* (FBO_b) contains several render targets. These attached textures have the same size $\frac{n}{4} \times n$ and internal float format, namely `GL_RGBA32F_ARB`, to avoid clamping and rounding. The minification and magnification filters are set to `GL_NEAREST` to prevent from distortion due to interpolation. Rendering has to be set up adequately to ensure pixel-to-texel mapping.

We need several render targets to store intermediate results and to adopt so-called ping pong rendering, i.e. alternately use two buffers as source and destination and vice versa. Note, that it is not possible to attach textures with different sizes or formats as render targets to one FBO. Therefore, at least two different offscreen buffers are required, one for the intensity content (FBO_a) and one for the packed RGBA content (FBO_b).

We have applied the basic idea to the computation of the Gradient Vector Flow (GVF) [XP97] as well, which represents an alternative, but more sophisticated image force field for active contours. Due to the complexity of the method, a more detailed explanation would go beyond the scope of this paper and is given in [KB09]. He et al. [HK06] introduced a similar approach. However, they did not consider any tiling and thus, only studied images up to sizes of one megapixel.

Recently, Tatarchuk [Tat08] proposed a method for the implementation of the overall snakes algorithm on the GPU using a greedy algorithm for the purpose of energy minimization. This approach does not account for a tiling of the image as well. Furthermore, only simple gradient-based image forces are computed, instead of the GVF.

Algorithm 2: Tile-based image forces on GPU

```
Input  : Tile array $T$ of $p \times q$ textures
         Coordinates $(i, j)$ of the current grid cell
Output: Image forces of the current grid cell $\vec{F}_{\text{img}}(i,j) = \left(F^x_{\text{img}}(i,j), F^y_{\text{img}}(i,j)\right)^T$
1: Render $T_{ij}$ into offscreen buffer having the size $n \times n$
2: Use pixel buffer object to reinterpret intensity frame buffer content as RGBA and
   generate packed texture $T'_{ij}$ of size $\frac{n}{4} \times n$
3: Normalize tile intensities by histogram stretching based on global intensity
   minimum and maximum
4: Enhance tile contrast
5: Filter tile with $5 \times 5$ Gaussian using separated kernels
6: Compute potential field (according to image energy definition)
7: Compute gradient of the potential field using two separate destination frame buffers
8: Read back pixel content of frame buffers into allocated main memory for the
   storage of $F^x_{\text{img}}(i,j)$ and $F^y_{\text{img}}(i,j)$
```

5. Results

Our algorithm was successfully applied for the semiautomatic contour extraction in several real images of histological cross-sections of the University of Göttingen, Germany, having sizes of up to 200 megapixels. All results were achieved on an Intel Core2 Quadcore Q6600 2.4GHz processor with 3GB RAM and an NVIDIA GeForce 8800 GTX.

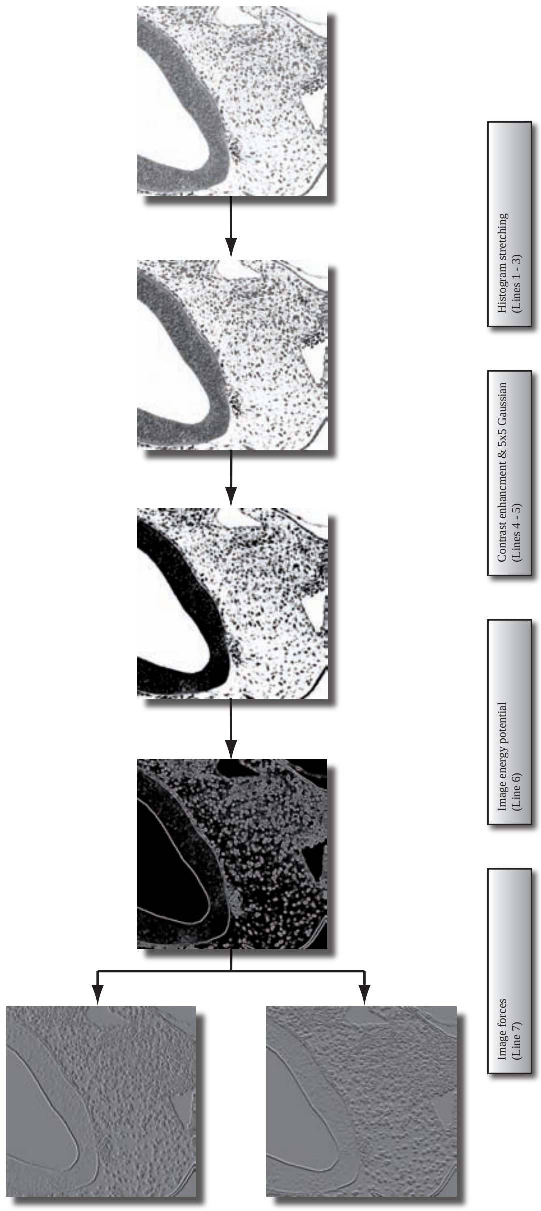
Histogram stretching (Lines 1 - 3)
Contrast enhancment & 5x5 Gaussian (Lines 4 - 5)
Image energy potential (Line 6)
Image forces (Line 7)

Figure 1: *Pipeline for image force computation on GPU (cf. algorithm 2). The two bottom tiles show the x (left) and the y component (right) of the final image force field.*

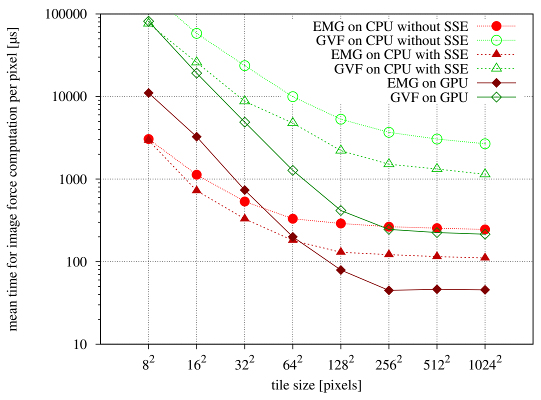

Figure 2: *Mean time per single pixel for different image force computations including image preprocessing for different tile sizes. Note, that the time scale is logarithmic.*

We have implemented the proposed computation scheme for traditional image forces (EMG) and the GVF on both CPU and GPU. Figure 2 compares computation times for different tile sizes n. Choosing $n = 256$ seems reasonable as speedup factors of up to 6 (EMG) and 15 (GVF) could be achieved with our new GPU-based approach.

Precomputing traditional image forces for the entire image area of a real sample image of 7150×6438 pixels takes about 6 seconds without GPU support. Instead, the initial image force computation on demand for an active contour, exploring 14% of the image tiles ($n = 256$), requires only 0.3 seconds with GPU support.

Tatarchuk's full GPU implementation [Tat08] is reported to run with at least 100 fps for image sizes of about 512×512 and 60 to 80 control points using an ATI Radeon HD 2900 XT. Comparing the results, we could outperform the method in spite of image force computations on demand and a CPU-based energy minimization using the calculus of variations and considering optimizations explained in [KVB06]. In fact, we computed about 500 deformation steps per second for the given sample image and approximately 1500 control points using our approach, emphasized in Figure 3.

6. Conclusion

In this paper, we presented a method to facilitate semiautomatic contour extraction on the basis of active contours in particular for huge images. It was shown, how a tiling approach enhances the snakes algorithm in different ways.

Both memory and computational effort could be significantly reduced as a result of local image force computations on demand, instead of a global precomputation upon the entire image domain. This approach made it possible to efficiently use snakes for contour detection in particularly big images. Moreover, we demonstrated how programmable

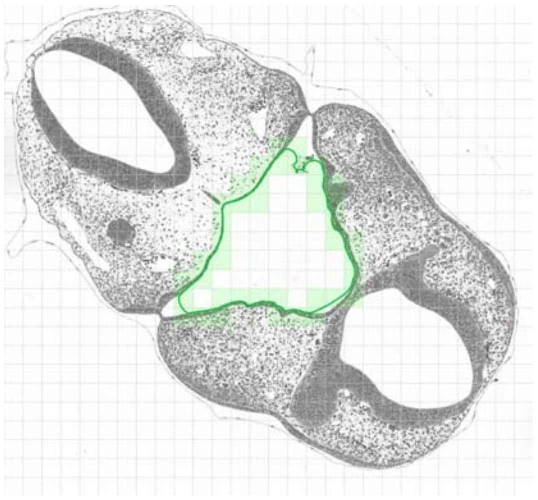

Figure 3: *Image with 7150 × 6438 pixels of a histological cross-section of "Tupaia belangeri" showing an intermediate state of a deforming active contour (green) with 1535 control points. Image forces are computed only for highlighted tiles.*

graphics hardware can be leveraged to further speedup the computation in order to compensate possible delay effects during the snake deformation. Finally, the decomposition of the input image into small tiles allows for a fast hardware based visualization by avoiding problems associated with texture size constraints.

References

[HK06] He Z., Kuester F.: GPU-Based Active Contour Segmentation Using Gradient Vector Flow. *Lecture Notes in Computer Science 4291* (2006), 191–201.

[KB09] Kienel E., Brunnett G.: *GPU-Accelerated Contour Extraction on Large Images Using Snakes.* Tech. Rep. CSR-09-02, Chemnitz University of Technology, 2009.

[KVB06] Kienel E., Vančo M., Brunnett G.: Speeding Up Snakes. INSTICC Press, pp. 323–330.

[KVK*07] Kienel E., Vančo M., Kowalski T., ClaussR., Brunnett G.: A Framework for the Visualization of Cross Sectional Data in Biomedical Research. Springer, pp. 77–97.

[KWT87] Kass M., Witkin A., Terzopoulos D.: Snakes: Active contour models. *International Journal of Computer Vision 1*, 4 (1987), 321–331.

[SWKK02] Süss M., Washausen S., Kuhn H.-J., Knabe W.: High resolution scanning and three-dimensional reconstruction of cellular events in large objects during brain development. *Journal of Neuroscience Methods 113* (2002), 147–158.

[Tat08] Tatarchuk N.: GPU-Based Active Contours for Real-Time Object Tracking. In *Shader X6: Advanced Rendering Techniques*, Engel W., (Ed.), vol. 1. Charles River Media, 2008, pp. 145–160.

[XP97] Xu C., Prince J. L.: Gradient Vector Flow: A New External Force for Snakes. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)* (1997), IEEE Computer Society, pp. 66–71.