GPU Programming in Computer Vision: Day 4

Date: Fri, 7 March 2014

Please work in groups of 2–3 people. We will check your solutions tomorrow after the lecture. Please be prepared to present your solution and explain the code. The general code requirements from exercise sheet 1 still apply. The bonus exercises are not mandatory.

Exercise 13: Denoising (8P)

Output: same number of channels as input image. Input: general number of channels.

Compute a minimizer u of the *denoising energy* with Huber-regularization,

$$\min_u \int_\Omega (u-f)^2 + \lambda\, h_\varepsilon\big(|\nabla u|\big)\, dx, \qquad \text{with} \quad h_\varepsilon(s) := \begin{cases} \frac{s^2}{2\varepsilon} & \text{if } s < \epsilon \\ s - \frac{\varepsilon}{2} & \text{else} \end{cases},$$

by solving the corresponding Euler-Lagrange equation:

$$2(u-f) - \lambda \operatorname{div}\Big(\widehat{g}\big(|\nabla u|\big)\nabla u\Big) = 0, \qquad \widehat{g}(s) := h'_\varepsilon(s)/s = \frac{1}{\max(\varepsilon, s)}.$$

1. Add Gaussian noise to your input image, with some noise variance $\sigma > 0$. For this, right after loading the input image, use the `addNoise` function from `aux.h`. You can set the noise level to e.g. $\sigma = 0.1$.
2. Compute the minimizer u by implementing the *Jacobi method* in several steps:
   (a) Compute $g = \widehat{g}(|\nabla u|)$. Reuse your code from exercise 11.
   (b) Compute the update step for u using the discretization from the lecture.
   (c) Compute N iterations and visualize the result. Test with different values λ. Check experimentally how many interations you need to achieve convergence (i.e. until visually there are no significant changes anymore).
3. Compute the minimizer u by implementing the *red-black SOR method* in several steps:
   (a) Compute g as above, and compute update step for u.

   The red and black updates are essentially the same, so write *only one* kernel, which will compute either the red-update or the black-update depending on a parameter. Note that you now also have a parameter $0 \le \theta < 1$ for the SOR-extrapolation.

   Hint: The SOR update step is essentially the same as for Jacobi. The only change is that there is an additional θ-extrapolation, and that the update is performed only in certain pixels.
   (b) Compute N iterations and visualize the result. Test with different λ values, and also with different values of $0 \le \theta < 1$.
   (c) Check how many interations you need for convergence. Do you observe a speed up when using SOR, compared to the Jacobi method?

Exercise 14: Inpainting – Gradient descent (5P)

Output: color. Input: color.

Compute a minimizer u of the *inpainting energy* with Huber-regularization,

$$\min_u \int_\Omega h_\varepsilon\big(|\nabla u|\big)\, dx, \qquad \text{with} \quad h_\varepsilon(s) := \begin{cases} \frac{s^2}{2\varepsilon} & \text{if } s < \epsilon \\ s - \frac{\varepsilon}{2} & \text{else} \end{cases}$$
$$\text{subject to } u = f \text{ in } \Omega \setminus A,$$

by gradient descent:

$$\partial_t u = \operatorname{div}\Big(\widehat{g}\big(|\nabla u|\big)\nabla u\Big) \quad \text{in } A, \qquad \widehat{g}(s) := h_\varepsilon'(s)/s = \frac{1}{\max(\varepsilon, s)},$$
$$u = f \quad \text{in } \Omega \setminus A.$$

1. Compute the inpainting mask $m : \Omega \to \{\texttt{true}, \texttt{false}\}$ which tells for each pixel (x, y) whether it is to be inpainted, i.e. $(x, y) \in A$, or not. In pixels $(x, y) \in A$, set the initial values of u to $u(x, y) = (0.5, 0.5, 0.5)$.
   Assume that the inpainting region A is specified by the *green* pixels in the input image. Use a `bool` array for m, and write a kernel to compute m from the input image. Use `bird_inpaint.png` as your test input image (the original image without the green marks is `bird.png`).

2. Compute the update step for u.
   Hint: The gradient descent equation is *exactly the same* as the nonlinear diffusion from exercise 11, except that u is updated only within the inpainting region A. Other values of u are left unchainged. Reuse your code.

3. Compute N iterations and visualize the result. How many iterations are needed until the inpainting in the region A is complete?

4. Compare different regularizers for the energy:

   (a) Huber regularizer $h_\varepsilon(|\nabla u|)$, as currently implemented.
   (b) Quadratic regularizer $\frac{1}{2}|\nabla u|^2$. This means using $\widehat{g}(s) = 1$.

   Is there any difference?

Exercise 15 (Bonus): Inpainting – Euler-Lagrange equation (2P)

Output: color. Input: color.

Compute a minimizer u of the inpainting energy from exercise 14 by solving the corresponding Euler-Lagrange equation:

$$-\operatorname{div}\Big(\widehat{g}\big(|\nabla u|\big)\nabla u\Big) = 0 \quad \text{in } A$$
$$u = f \quad \text{in } \Omega \setminus A.$$

using the *red-black SOR scheme*. This should result in a much faster algorithm to compute the inpainting.

1. Reuse your code from exercises 13 and 14 to compute the inpainting mask m and an update step for u using the *red-black SOR scheme*.
   Hint: The SOR update scheme is almost the same as for the denoising case. The only difference is that the update happens only in $(x, y) \in A$, and the update formula has no terms for the input image f (i.e. no $2f$ in the numerator and no 2 in the denominator).

2. Compute N iterations and visualize the result. Compare the convergence speed of SOR and gradient descent (exercise 14) in terms of the number of needed iterations N until convergence. Do you observe a speed up when using SOR?