BSpeckleRender

A Boolean model for deformed speckle rendering

December 2017

Contents

1 Contents of the zip file **1**

2 How to run the Matlab codes? **3**
2.1 Set up Matlab environment 3
2.2 Run example scripts 4

These Matlab codes can be used for non-profit academic research only. They are distributed under the terms of the GNU general public license v3.

Codes have been tested under Matlab 2017a.

Anyone finding the sample images or the Matlab codes useful is kindly asked to cite:

F. Sur, B. Blaysat, and M. Grédiac.
Rendering deformed speckle images with a Boolean model.
Journal of Mathematical Imaging and Vision, 2018.

1 Contents of the zip file

Unzipping the archive `BSpeckleRender.zip` gives the following directories and files:

- `documentation.pdf` : the present file
- `classes/` : definition of Matlab classes
- `functions/` : Matlab functions for rendering speckle images

- `displacements/` : Matlab functions for defining mappings ϕ (or inverse displacement field), cf. Section 2.2 of the paper.

- `scripts/` : examples of Matlab scripts

Please read the `.m` files or the help of the functions (`help function_name`) for the description of the parameters. We stick to the notations of the paper.

Note that the functions in `displacements/` do not accept input parameters in order to run on GPU (it seems to be a current limitation of Matlab's `arrayfun`). It is mandatory to write a separate function for different values of the parameters, as it is done in `translation01.m` to `translation10.m`.

The list of the functions in the directory `functions/` is:

- `SpeckleRenderCPU` & `SpeckleRenderGPU.m`: CPU & GPU version of speckle image rendering, before gray-level quantization.

- `quantization`: quantize the output of `SpeckleRenderCPU` or `SpeckleRenderCPU` on b bits. Note that, if $b \leq 8$, the output image is a `uint8` array that can be saved as a 8-bit PNG (or TIF) file; if $8 < b \leq 16$ the output image is a `uint16` array that can be saved as a 16-bit PNG (or TIF) file. Note also that if $b \neq 8$ or if $b \neq 16$, the gray-level do not span the set of all possible values of 8-bit or 16-bit format. In order to get a consistent display (using `imshow`), the user may wish to multiply the gray-level by 2^{8-b} or 2^{16-b} in order to span $[0, 2^8]$ or $[0, 2^{16}]$, for instance:

  ```
  >> figure, imshow(2^4*Ima);
  ```

  for a 12-bit image array `Ima`.

- `display_displacement`: display the inverse displacement field U and its components U_x or U_y. See examples in `script_Fig3` or `script_modsinewave`.

- `addnoise`: add noise (either Gaussian white noise or signal-dependent noise) to an image. See Section 3.5 of the paper. This can be made by using:

  ```
  >> NoisyImage = addnoise(myparamnoise, b, Image);
  ```

  where `myparamnoise` is an object with noise parameters (see the class `paramnoise`) and `Image` is a noise-free image, coded either in `uint8` or in `uint16`. The output image `NoisyImage` has the same format as the input image and can be displayed with `imshow`.

  For instance, noise parameters as in the Sensicam QE camera (this is valid only for 12-bit images) would be set by:

```
>> myparamnoise=paramnoise('S',0.519,1.33-0.519*45.46);
```

since the datasheet of this camera indicates: $\eta^2 = 1.33$, $g = 0.519$, $\nu = 45.46$.

Note that this is only an approximation of a realistic Poisson-Gaussian signal-dependent noise.

See also script `script_addnoise`.

- `computeintensityCPU` / `computintensityGPU`: CPU / GPU version of the Monte Carlo estimation $\widehat{I}$, (cf. line 9 in Algorithm 1 of the paper), called by `SpeckleRenderCPU` and `SpeckleRenderGPU.m`, respectively.

- `computeintensitiesGPU`: called by `computeintensityGPU`.

When generating the user's own speckle images, he/she should use a low bit-depth (e.g., 4) when playing with sensor and speckle parameters, in order to limit the computation time.

2 How to run the Matlab codes?

2.1 Set up Matlab environment

1. Compile the mex file `computeintensityCP.c`:

   ```
   >> cd functions/
   >> mex computeintensityCPU.c
   >> cd ..
   ```

   If the user is not able to compile this file and in the absence of the corresponding compiled file, the slower `computeintensity.m` function is used.

2. Set up Matlab path:

   ```
   >> addpath('classes','displacements','functions','scripts')
   ```

3. Set up the GPU (if available):

   ```
   >> gpuDevice
   ```

   If the user is not able to use the GPU, the calls to the function `SpeckleRenderGPU` should be replaced by `SpeckleRenderCPU.m` in the scripts.

4. Set up parallel pool (if the CPU version of the code is used):

   ```
   >> parpool
   ```

2.2 Run example scripts

Several example scripts are provided to the user in `scripts`:

- Generate speckle images with the same parameters as Fig. 2 in the paper:

  ```
  >> script_Fig2
  ```

- Generate speckle images with the same parameters as Fig. 3 in the paper:

  ```
  >> script_Fig3
  ```

- Generate the results showed on Fig. 4 in the paper:

  ```
  >> script_Fig4
  ```

- Generate a sample of 10 translated speckle images (displacement step equal to 0.1 pixel):

  ```
  >> script_translation
  ```

  (see `script_translation.m` file for the parameters).
  Script used to generate `sample1.tgz`.

- Generate a reference image and a deformed image with a frequency modulated sine wave:

  ```
  >> script_modsinewave
  ```

  Script used to generate `sample2.tgz`.

- Generate a reference image and a deformed image with a zoom and rotation

  ```
  >> script_zoomrotation
  ```

- Usage of `addnoise` and `paramnoise`:

  ```
  >> script_addnoise
  ```