ThreeB

1.1

Generated by Doxygen 1.7.4

Wed Nov 28 2012 15:39:12

Contents

1 3B Microscopy Analysis

1.1 Introduction

This project contains the reference implementation of the 3B microscopy analysis method, and an ImageJ plugin. Please refer to `the project website` for more information on the method.

To get started with analysing data, the `ImageJ` plugin is the most suitable piece of software. This can be obtained from the `the project website`.

For more advanced analysis (such as running on a cluster), the the program `multispot5_-headless` should be used. This program needs to be run from the commandline.

This project contains the source code for the commandline program and the ImageJ plugin.

1.2 Getting started

1.2.1 Dataset types and experimental parameters

Bayesian analysis of blinking and bleaching allows data to be extracted from datasets in which multiple fluorophores are overlapping in each frame. It can of course also be used to analyse standard localisation (PALM/STORM) data, but it must be borne in mind that the low density and high frame number of such datasets can lead to long runtimes. Here we briefly discuss the different types of dataset, related to different applications, that you may wish to analyse using 3B.

1.2.1.1 Low density PALM/STORM datasets

These datasets have few fluorophores overlapping in each image and are at least 10,000 frames long. As discussed above, they can be analysed with 3B but their run time makes this a large time investment. If you wish to use this approach for performance verification, we would suggest selecting a small spatial area (around 1.5-3 μm square). The algorithm can also be parallelised by running different sets of a few hundred frames for the same area separately. Even with parallelisation, the large number of frames will make it time consuming to run. If you simply want to know what the structure is like, we would suggest using a method such as QuickPALM, which are very fast in comparison.

1.2.1.2 High density fixed cell datasets

These datasets are of fixed cells but have multiple fluorophores overlapping in each frame. You may acquire this type of dataset if the system you use has a fluorophore which cannot be photoswitched, or the blinking properties of which cannot be tuned over a wide enough range using the embedding medium, or if your light source is not powerful enough to drive most of the fluorophores into the non-emitting state. In fixed samples labelled with fluorescent proteins there will in our experience be almost no blinking present, and 3B will therefore pick up bleaching. While this can produce satisfactory results, the localisation is on single event on a high background and therefore the performance will be significantly degraded.

Some users choose to acquire this type of dense data if they have severe problems with drift, particularly in the z-direction, as it drastically cuts the drift over the acquisition time. However, in the long term it is worth trying to improve the stability of such systems, since z-drift will impact the accuracy of all types of localisation measurement. If you are unsure how badly your system is affected by z-drift, it is useful to carry out a calibration of the drift of the microscope using a bead sample before starting acquisition of data for superresolution.

1.2.1.3 Live cell datasets

These datasets are of live cells, generally labelled with standard fluorescent proteins such as mCherry. The mounting medium should be phenol red free, to avoid unnecessary background. The intensity of the light source should be selected such that it is high

enough to produce blinking but not strong enough to completely bleach the sample over the time of the acquisition. For example, for a standard Xe arc lamp illumination with the full power of the lamp is generally suitable, but if you are using a powerful laser source it is recommended to take multiple datasets with different power levels to determine which is suitable.

The time which is needed to acquire the data necessary for a single superresolution frame is dependent on the illumination intensity, the speed of the camera, and the properties of the flurophore. Many older EMCCD cameras have a maximum acquisition speed of around 50 frames per second for a small region of interest, which then gives a limit of 4 s to acquire 200 frames. High specification EMCCD cameras and SCMOS cameras have much higher acquisition speeds of up to thousands of frames per second for restricted areas. However, in order to maintain the number of photons from each flurophore per frame the illumination intensity would have to be increased, which is likely to bleach the sample rapidly, and change the blinking properties of the fluorophore to some extent.

The acquisition of live cell datasets allows dynamics to be observed. It should be noted, however, that if the structures move over the timescale of the acquisition then that movement will cause blurring in the reconstructed image.

1.2.1.4 Live cell datasets

The 3B algorithm must be able to pick up the changes in intensity which occur when a fluorophore switches between an emitting and a non-emitting state. The accuracy with which localisation can occur depends, as for other localisation techniques, on the number of photons from the fluorophore and the background level. Since 3B is generally used with a widefield setup, if the sample has a lot of fluorescent structure out of the plane of focus, the background will be higher and it will be more difficult to localise. For thick samples, the background can be reduced by using TIRF or high angle illumination.

1.2.2 Iterations and run time

In general, determining the number of iterations required for MCMC algorithms is an unsolved problem. A good general rule is that once the reconstruction stops changing significantly with increasing iterations, then it is likely that the reconstruction has converged to a reasonable point.

If you are unsure, then rerun exactly the same area with exactly the same parameters, but with a different random seed. Note that the ImageJ plugin will automatically select a different seed each time in the standard interface, but with the advanced interface or commandline program, the seed must be specified in the configuration file (see The configuration file and advanced settings). If the two results appear essentially the same, then it is very likely that a sufficient number of iterations has been reached.

A good general rule is that 200 iterations is sufficient for convergence under almost all circumstances. For the example usage given in this manual, the required number of iterations for good convergence requires about 6 hours on a standard PC (Core i7 at 3GHz).

Convergence may be achieved before 200 iterations, but terminating the run before convergence can lead to artefacts. As with any microscopy method, the experiment and

analysis should always be carried out appropriately to minimise the risk of artefacts.

There are a number of issues which can lead to artefacts:

- Early termination of the algorithm
  - The algorithm builds up a reconstrutcion using a number of random samples. If the algorithm is terminated too early, then the reconstruction will be dominated by the randomness, and there will not be enough samples for random fluctuations to average out.
  - MCMC algorithms may exhibit a property called `burn in`. As the 3B algorithm runs, it maintains an estimate of the number of spots present in the image. Since it is an MCMC algorithm, this estimate will fluctuate around some mean value. However, 3B can add or remove spots at a rate of at most 5 per iteration (usually much slower). If the algorithm is started with a very bad estimage of the number of spots then it may take many iterations for it to approach the mean. During this time, the samples drawn will not be representative.

    3B must therefore run for enough iterations that the later, representative iterations will dominate and swamp the earlier ones. If 3B is started using a very bad estimate, then this can require substantially more than 200 iterations.

- Bright regions close to the boundary
  - The 3B algorithm will not examine any pixels outside the boundary of the marked region. If there is a bright region of the image on or near the boundary, then 3B will naturally attempt to places fluorophores there. However since the point spread function of the microscope is typically several pixels across, the images of fluorophores near the boundary will extend across it and so will be missing information, which could lead to artefacts.

    If possible, placing the boundary near a bright region should be avoided. If this is not possible, then the reconstruction close to the boundary should be ignored. Anything further than the PSF diameter from the boundary is unlikely to be affected. Typically this is around 3 pixels.

- Insufficient background regions
  - The 3B algorithm needs to be able to estimate the babkground noise level in order to model the image correctly. If there is an insufficient amount of background (for instances if the images are too small), then the estimation of the noise level may be poor which could lead to artefacts.

    The sample data provided is an example of data for which the background can be estimated effectively.

- Image drift or motion
  - The 3B algorithm does not model image motion.

    If there is significant motion for example due to drift or a live cell moving then artefacts may result. The artefacts take the form of streaking or smearing in

the direction of drift, or structure bunching randomly at one end of the drift or the other.

Ideally the experiment should be run to minimize drift, but if this is not possible, then drift correction software (for example based on tracking beads) should be used to correct the drift. For live cell analysis, a tradeoff can be made between spatial and temporal resolution. If fewer frames are analysed, the temporal resolution is higher but the spatial resolution will be degraded. Varying the number of frames analysed can also be used to investigate the impact of sample movement in live cells, if this is a concern.

- Incorrect parameters
  - If the parameters (especially the FWHM of the point spread function of the microscope) are set incorrectly then 3B will not be able to model the image correctly, which may lead to poorer resolution or artefacts. The FWHM can be readily determined by taking a diffraction limited image of beads.
- Very high background levels
  - See Live cell datasets .

1.3 Using the ImageJ Plugin

A tutorial for using the ImageJ plugin is provided under `Plugins>3B>Help`

The plugin can operate in two modes, standard and advanced. The standard mode allows the user to set the microscope PSF and spot size, the starting number of spots for the analysis and the range of frames.

The plugin also offers an advanced mode of operation which allows much greater control over 3B. See The configuration file and advanced settings for further details.

1.4 Using the commandline program

We have provided a set of test data on the website. Download and unpack the zip file. It will create a new directory called test data with the collowing contents:

```
test_data/AVG_test_data.bmp
test_data/markup1.bmp
img_000000000.fits
img_000000001.fits
img_000000002.fits
...
img_000000299.fits
```

Then run the following command:

```
./multispot5_headless --save_spots test_data/results.txt --log_ratios test_data/m
      arkup1.bmp test_data/img_000000*
```

The program will save the results in the file `test_data/results.txt` . The program will run indefinitely in the default setup, but you may view the results at any stage. There is no well defined stopping point for this type of algorithm, so it is advisable continuously monitor the resultant image, and stop the algorithm when the output image is no longer changing with time. After 30 minutes on a fast PC (e.g. Core i7 975), the ring structure which is not resolved in the widefield image should be clearly visible. After about 75 mins, the finer details of the structure begin to approach those seen in Fig 2e in the associated paper.

The ImageJ plugin can load a results file and perform a reconstruction.

Alternatively, you can process the results file further in order to view the results. Run the following command:

```
awk '/PASS/{for(i=2; i <=NF; i+=4)print $(i+2), $(i+3)}' test_data/results.txt >
      test_data/coordinates.txt
```

The file `test_data/coordinates.txt` contains a long list of (x,y) coordinates, representing possible spots positions. In order to view the results, load the data into a graph plotting program and create a scatter plot. NOTE: the axes are in pixel coordinates, so you will have to multiple any distances by the number of nm/pixel in order to get distances in nm.

1.4.1 Example usage explained

1.4.1.1 Test Data

The 300 TIFF files in the test directory correspond to the data used for Fig. 2 in the paper. Please refer to the paper for details on how the data was obtained.

The file `AVG_test_data.bmp` is a Z projection made using `ImageJ`.

The file `markup1.bmp` is a mask indicating which area of the image to analyse. All perfectly black pixels are ignored, ecerything else is analysed. If you overlay `markup1.bmp` and `AVG_test_data.bmp` you can see which area the markup corresponds to. The markup file was created using `the GIMP`.

1.4.1.2 Running the program

The general form for running the program is:

```
./multispot5_headless [ --variable1 value1 [ --variable2 value 2 [ ... ] ] ] imag
      e1 image2 ...
```

so the example sets ths variable `save_spots` to `test_data/results.txt` and the variable `log_ratios` to `test_data/markup1.bmp`. The remaining arguments is the list of files to be analysed.

The program gets the markup in the filename given in the `log_ratios` variable (yes, the choice of name is very strange, and corresponds to a very old phase of development). The more sanely named variable `save_spots` is the filename in which the output is to be saved.

The program actually has a large number of variables which must be set. Most of them you probably don't want to change, but some of them you will want to change. The default values for these variables are stored in `multispot5.cfg` The format of this file should be mostly self explanatory. Everything after a `is` a comment and is ignored. See The configuration file and advanced settings for further details.

You will probably want to change:

- `blur.mu`

  This is the prior over spot size, which is how the pixel size and microscope FWHM are represented. Some example values for a FWHM of 300nm/pix at 160 and 100 nm/pix and for a FWHM of 270nm at 79nm per pixel.

  If you have significantly largre or smaller pixels, the performance may be degraded.

- `placement.uniform.num_spots`

  This is the initial number of spots to be placed down. Eventually, the algorithm will converge to a reasonable number of spots, even if this value is far off. The default value (15) is appropriate given the small area and dimness of the sample data. You will want to increase this number for larger areas of markup and relatively brighter regions.

  If this number is more than 1000, then the algorithm will run very slowly and may take several days.

Note that variables specified on the commandline override all variables in the configuration file.

The program can read FITS, BMP, PPM and PGM images. Depending on how it has been compiled, it can also read TIFF, PNG and JPEG images. The program cannot work on multi-image TIFF files. ImageJ can be used to split a multi-image TIFF into a collection of single image files. All the images loaded must be the same size.

1.4.1.3 Extracting and visualising the data

The output file (in this case `test_data/results.txt`) containing the results is in a format unsuitable for plotting directly, and must be extracted. The reason for this is that the output file contains enough information to seamlessly continue long runs which have been interrupted. The provided AWK program extracts the coordinates of the spots over all iterations and puts them in `coordinates.txt`.

Alternatively, the data can ve visualised using the plugin under the menu `Plugins>3B>Open 3B run`

1.5 The configuration file and advanced settings

The 3B system has a large number of parameters which control its behaviour. These are controled via the configuration file for the commandlie program or via the "Advanced" option for the plugin. The "Advanced" option essentially allows you to supply a fully custom configuration file.

The sample configuration file is given below along with explanations of all parameters. In the program, most of these parameters are used by the FitSpots class.

```
////////////////////////////////////////////////////////////////////////////
//
// Advanced setting.
//
// You probably do not want to change most of these.
//
// Parameters which you are most likely to need to change are marked
// CHANGE THIS
//
// Note that if a variable is set twice, the last setting takes precedence.
//
// Comments in this section should be read in conjunction with the paper,
// online methods and supplementary material.
//
// Any line starting with // is a comment and is ignored by the
// plugin.


////////////////////////////////////////////////////////////////////////////
//
// Things you might want to change are below here
//

//Random seed for the Monte-Carlo process.
//Changing this value will change the random numbers which are generated.
//Please refer to the user manual section "Iterations and run time" for
//further discussion.
seed=0

/////////////////////////////////////////
//
// Spot intensity priors. You can probably leave this as-is unless your
// fluorophores are very much brighter compared to the noise than the ones
// in the paper. See "Modelling the image" in Supplementary Note 1.
//
// See here for definitions of mu and sigma:
// http://en.wikipedia.org/wiki/Log-normal_distribution
//
// Note that these values are for the spot brightness once the image
// has been normalised to have unit standard deviation so these should
// not be changed unless your images are very significantly less noisy
// than the data presented in the paper.
//
intensity.rel_mu=2
intensity.rel_sigma=1


/////////////////////////////////////////
//
// Spot shape priors. CHANGE THIS
//
// This specifies the log-normal distribution for the prior over spot
// radius. This is specified in pixels, not nm.

//Corresponds to 300nm at 100nm per pixel
blur.mu=0.242
blur.sigma=0.1

//Corresponds to 300nm at 160nm per pixel
```

```
blur.mu=-0.218
blur.sigma=0.1

//Corresponds to 270nm at 79nm per pixel
//blur.mu=0.37261
//blur.sigma=0.1


//Number of spots. CHANGE THIS
//
//15 is suitable for a small image patch, such as the example in the
//help menu.
//
//If you set this too high, the system will run very slowly.
placement.uniform.num_spots=15



////////////////////////////////////////////////////////////////////////////
//
// You probably do not want to change anything below here.
//


//Transition matrix and initial probabilities

// Following the notation in [24], A represents the transition
// probabilities of the HMM and pi the initial state. A corresponds
// to Supplementary Figure 4 as:
//    A = [ P1 P2 0; P3 P4 P5; 0 0 1]
// These are used in the inner integral of eq 7.

A=[0.16 0.84 0; 0.495 0.495 0.01; 0 0 1]
pi=[.5 .5 0]



////////////////////////////////////////////////
//
//General parameters

// All MCMC systems can use mixing samples[28], which are drawn by
// by the sampler but do not take part in further computations.
// For every mixing_iterations samples drawn 1 is uses, so
// mixing_iterations=1 corresponds to no mixing.

//Number of mixing iterations used by all Gibbs samplers in this algorithm.
gibbs.mixing_iterations=1



////////////////////////////////////////////////
//
//Specific subsystem parameters


////////////////////////////////////////
//
// Main optimization - step 2 of the algorithm

//maximum l_inf of motion vector during optimization stage.
//See step 2 of the algorithm,
```

```
cg.max_motion=0.5

//Scale the maximum step in brightness according to the prior shape
//The reason is that the brightness is a very different scale from
//the spot position and shape, so the optimizer will get stuck
//taking many tiny steps in the direction of brightness.
//
//If this is set to 0 then no scaling is used.
//
//If it is set to 1, then scale by the brighness step by the standard
//deviation of the prior distribution.
max_motion.use_brightness_std=1

//Maximum number of iterations of conjugate gradient optimization
//See [29] for a description of an iteration in this context.
main.cg.max_iterations=5

//Number of samples used for computing gradient using hybrid method.
//This corresponds to S, as applied to step 2 of the algorithm.
main.gibbs.samples=10

//Number of passes of optimization in step 2 to be completed before
//advancing to step 3.
main.passes=4


//Allow spots to be at most this far outside the marked region
//Anything further outside will be removed automatically.
position.extra_radius=2.3

////////////////////////////////////////
//
// Add/remove spots - step 3 of the algorithm

//When computing Z, in step 3a and 3c, use a uniform position prior (to
//make spot probability a proper distribution). If this is set to 0, then
//no position prior is used. Do not set to 0 except for debugging.
position.use_prior=1

//Number of samples used to compute the gradient usint the
//hybrid method, as applied to step 3c.
add_remove.optimizer.samples=20

//The optimizer can get stuck on saddle points. If a saddle point
//is found, the the optimizer will restart in the direction corresponding
//to the worst eigen value. Up to this many restarts will be attempted before
//gving up. Step 3c.
add_remove.optimizer.attempts=10

//There is no efficient algorithm for evaluating the Hessian (equivalent
//to the inner integral in Eq 6, but for second derivatives). The integral
//is approximated using Monte-Carlo integration (note, not MCMC), with this
//many samples drawn from the Markov chain using FFBS.
//
//The outer summation uses add_remove.optimizer.samples samples.
add_remove.optimizer.hessian_inner_samples=1000

//Number of cycles of thermodynamic integration  (step 3a and 3c) and
//eq 2.
//Note that the equation 1.25^(i/10) only applies if this number is set
//to 1000 as it was for all experiemnts in the paper. The actual equation is:
//1.25^(100 * i/add_remove.thermo.samples)
```

```
add_remove.thermo.samples=1000

//Outer and inner samples for computing the Hessian using the hybrid method
//in step 3e.
//
//See add_remove.optimizer.hessian_inner_samples for an explanation of the
//meaning.
add_remove.hessian.outer_samples=100
add_remove.hessian.inner_samples=1000

//Number of repeats of step 3.
add_remove.tries=10


////////////////////////////////////////////////
//
// Image preprocessing.
//
// See "Modelling the image" in supplementary methods.
// The image is normalized to locally zero mean and globally unit standard
// deviation.

// Skip the mean removal step if this is set to 1.
// This is really only for debugging. Do not change.
preprocess.skip=0

//Low pass filter used for mean removal has has this sigma
preprocess.lpf=5



////////////////////////////////////////////////
//
// Initial spot placement.
//

//Options are:
//  uniform (place spots uniformly over the area of interest)
//  intensity_sampled  (place spots more densely in brighter regions)
//intensity_sampled generally leads to slightly faster convergence.
placement=intensity_sampled



////////////////////////////////////////////////////////////////////////////
//
// Things after here do not do anything.
//
// Things after here have no effect in the plugin, but are required if
// this config file is to be used in the standalone program
//

//Instead of calculating the scaling of the initial image to set the
//standard deviation to 1, use a fixed scaling. This is for debugging
//purposes and is very unlikely to be useful otherwise.
preprocess.fixed_scaling=0

//Threshold the log_ratios image at this level in order to find
//regions for analysis.
threshold=0

//Dilate regions found by thresholding by this radius.
```

```
radius=0

//Analyse marked region number 19:
cluster_to_show=19

//If this is set to 1, cluster_to_show is ignored and the largest
//region is used instead.
use_largest=1
```

2 Compiling the programs

In order to compile the project, you will need to download and install the following libraries:

- TooN http://www.edwardrosten.com/cvd/toon.html
- libcvd http://www.edwardrosten.com/cvd/
- gvars3 http://www.edwardrosten.com/cvd/gvars3.html

The program is portable and is well tested under Linux and OSX. It will also compile under Windows using cygwin or MinGW.

The program can be built using the usual method for compiling under Linux:

```
./configure && make
```

2.1 Compiling the ImageJ plugin

The plugin is provided pre-compiled from the project website.

There are two ways of building the plugin, manual and automatic. If you want to make changes to the plugin, then use manual building. If you want to automatically build the plugin for several platforms, then use automatic building.

2.1.1 Manual

The basic build instructions are the same as for the commandline program. You will also need the JDK (Java Development Kit) and ImageJ installed.

You will have to locate where your system has installed the JDK. If it is not in /usr/lib/jvm/java-6-openjdk/include, you will have to specify the path:

First configure the system:

```
./configure --with-imagej=/path/to/ImageJ/ij.jar --with-jni=/path/to/jdk/include
```

You will also need to make sure that the JDK programs (javac, havah, etc) are in your path. The configure script will attempt to detect the location of the JNI headers. If it fails, you will need to specify `--with-jni=/path/to/jdk/include`

To build the JAVA part:

```
make three_B.jar
```

To build the plugin (on Linux):

```
make libthreeB_jni.so DYNAMIC_PLUGIN=1
```

Note that if you do not specify `DYNAMIC_PLUGIN`, then the makefile will try to build a plugin with some dependencies statically linked in which will almost certainly fail unless you have set the system up to support such an operation.

On MinGW:

```
make threeB_jni.dll
```

Now copy three_B.jar and libthreeB_jni.so into your ImageJ plugins directory.

2.1.2 Automatic Build

The automatic build method is very slow and is designed to be able to repeatably build plugins for 32 and 64 bit Linux and Windows. It is also designed to build the plugin with as many static dependencies as possible so that only a single DLL/so needs to be shipped per system.

The script operates by building a temporary install of Ubuntu 10.04 LTS, and using that to compile all variants of the plugin.

You will need a Debian based system (or a system on which the command `debootstrap` works) and root access.

Tha automatic build system makes use of cLAPACK, rather than LAPACK as the LAPACK part is not speed critical and it is easier to build CLAPACK without additional external dependencies.

To build, run the following commands:

```
#First make a tar.gz of the source code
bash make_dist.sh

#Now execute the automatic build process
bash build_plugin.sh
```

The build takes a long time, and you should probably edit `build_plugin.sh` to point the installer at an Ubuntu mirror somewhere near to where you are.

At the end of the build, the script will print out a directory name like:

```
dist-123908
```

A fresh copy of the plugin DLL and shared object will be present in that directory named.

3 Module Index

3.1 Modules

Here is a list of all modules:

4 Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

5 Class Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

6 Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

7 File Index

7.1 File List

Here is a list of all files with brief descriptions:

8 Module Documentation

8.1 Useful debugging functions.

Functions

- Image< byte > scale_to_bytes (const Image< float > &im, float lo, float hi)
- void test_output_patch_variance (const vector< Image< float > > &ims)
- template<class C >
  void assert_same_size (const C &images)

8.1.1 Function Documentation

8.1.1.1 Image<byte> scale_to_bytes (const Image< float > & *im*, float *lo*, float *hi*)

Scales an image in to the correct range for bytes.

Parameters

hi	Brightest pixel in the image
lo	Dimmest pixel in the image
im	Image to scale

Returns

scaled image

Definition at line 9 of file debug.cc.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes(), test_output_-patch_variance(), and FitSpots::try_modifying_model().

```
{
    Image<byte> out(im.size());
    for(int r=0; r < out.size().y-0; r++)
        for(int c=0; c < out.size().x-0; c++)
            out[r][c] = (int)floor((im[r][c]-lo)*255/(hi-lo));

    return out;
}
```

8.1.1.2 void test_output_patch_variance (const vector< Image< float > > & *ims*)

Find the variance of every patch in the image and save it to a file.

Parameters

ims	List of images.

Definition at line 23 of file debug.cc.

References assert_same_size(), scale_to_bytes(), and sub_images().

```
{
    assert_same_size(ims);

    int rr = GV3::get<int>("test.variance.radius", 1, -1);
    ImageRef r(rr, rr);
    ImageRef size = r*2 + ImageRef(1,1);

    Image<float> stds(ims.front().size(), 0);

    ImageRef p;
    for(ImageRef p(0,0); p.y < stds.size().y - size.y; p.y++)
    {
        for(p.x=0; p.x < stds.size().x - size.x; p.x++)
            stds[p + r] = sqrt(mean_and_variance(sub_images(ims, p, size)).second
      );
    }

    SubImage<float> s = stds.sub_image(ImageRef(2,2), stds.size() - ImageRef(4,4)
      );

    float hi = *max_element(s.begin(), s.end());
    float lo = *min_element(s.begin(), s.end());
    cerr << hi << " " << lo << endl;
    img_save(scale_to_bytes(stds, lo, hi), "test_variance.png");
}
```

8.1.1.3 template<class C > void assert_same_size (const C & *images*)

Determines that all images in the incoming container are the same size, and that the container is not empty.

Parameters

images	Container to check

Definition at line 11 of file debug.h.

Referenced by auto_fixed_scaling(), average_image(), FitSpots::FitSpots(), generate_state_parameters_ye_olde(), SampledMultispot::GibbsSampler::GibbsSampler(), SampledMultispot::GibbsSampler2::GibbsSampler2(), and test_output_patch_variance().

```
{
    assert(!images.empty());
    for(typename C::const_iterator i=images.begin(); i != images.end(); i++)
        assert(i->size() == images.front().size());
}
```

8.2 General utility functions.

Classes

- struct MT19937

Useful wrapper for MT19937 random number generator class.

- class Kahan

  Class implementing the Kahan summation algorithm to allow accurate summation of very large numbers of doubles.

Typedefs

- typedef std::pair< CVD::ImageRef, CVD::ImageRef > BBox

Functions

- double ln (double x)
- Vector spots_to_Vector (const vector< Vector< 4 > > &s)
- vector< Vector< 4 > > spots_to_vector (const Vector<> &s)
- Image< byte > scale_to_bytes (const Image< float > &im, float lo, float hi)
- Image< byte > scale_to_bytes (const Image< float > &im)
- Image< float > average_image (const vector< Image< float > > &ims)
- vector< string > split (const string &line)
- template<class C >
  string xtoa (const C &x)
- template<class C >
  C atox (const string &s, const string &msg)
- set< ImageRef > dilate_mask (const vector< ImageRef > &v, double r)
- vector< int > SampledMultispot::sequence (int n)
- double sign (double x)
- float sq (float f)
- const std::vector< CVD::SubImage< float > > sub_images (const std::vector< CVD::Image< float > > &im, CVD::ImageRef pos, CVD::ImageRef size)
- std::pair< CVD::ImageRef, CVD::ImageRef > boundingbox (const std::vector< CVD::ImageRef > &all_spots)
- template<class Stream >
  void open_or_die (Stream &save_spots, const std::string &save_spots_file)

8.2.1 Typedef Documentation

8.2.1.1 typedef std::pair<CVD::ImageRef, CVD::ImageRef> BBox

Deffinition of a pixel aligned bounding box.

Definition at line 49 of file utility.h.

8.2.2 Function Documentation

8.2.2.1 double ln (double *x*) [inline]

Computes the natural logarithm, but returns -1e100 instead of inf for an input of 0.

This prevents trapping of FPU exceptions.

Parameters

x	*x*

Returns

ln *x*

Definition at line 15 of file forward_algorithm.h.

Referenced by backward_sampling(), diff_log_log_normal(), forward_algorithm(), forward_algorithm_delta(), forward_algorithm_delta2(), forward_algorithm_hessian(), forward_backward_algorithm(), hess_log_log_normal(), log_log_normal(), and FitSpots::try_modifying_model().

```
{
    if(x == 0)
        return -1e100;
    else
        return std::log(x);
}
```

8.2.2.2 Vector spots_to_Vector (const vector< Vector< 4 > > & *s*)

There are two sensible ways of storing the state vector of spot positions.

This function converts between them. See also spots_to_vector.

Parameters

s	list of spots to convert

Definition at line 99 of file multispot5.cc.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes(), FitSpots::run(), and FitSpots::try_modifying_model().

```
{
    Vector<> r(s.size()*4);
    for(unsigned int i=0; i < s.size(); i++)
    {
        r[i*4+0] = s[i][0];
        r[i*4+1] = s[i][1];
        r[i*4+2] = s[i][2];
        r[i*4+3] = s[i][3];
    }
    return r;
}
```

8.2.2.3 vector<Vector<4> > spots_to_vector (const Vector<> & *s*)

There are two sensible ways of storing the state vector of spot positions.

This function converts between them. See also spots_to_Vector.

Parameters

s	list of spots to convert

Definition at line 116 of file multispot5.cc.

Referenced by NegativeFreeEnergy::compute_with_mask(), generate_state_parameters_-ye_olde(), and NegativeFreeEnergy::operator()().

```
{
    vector<Vector<4> > r(s.size()/4);
    for(unsigned int i=0; i < r.size(); i++)
        r[i] = s.slice<Dynamic, 4>(i*4, 4);
    return r;
}
```

8.2.2.4 Image<byte> scale_to_bytes (const Image< float > & *im,* float *lo,* float *hi*)

Normalize an image for display purposes.

Definition at line 126 of file multispot5.cc.

```
{
        Image<byte> out(im.size());
        for(int r=0; r < out.size().y-0; r++)
                for(int c=0; c < out.size().x-0; c++)
                        out[r][c] = (int)floor((im[r][c]-lo)*255/(hi-lo));
        return out;
}
```

8.2.2.5 Image<byte> scale_to_bytes (const Image< float > & *im*)

Normalize an image for display purposes.

Definition at line 137 of file multispot5.cc.

```
{
        float lo = *min_element(im.begin(), im.end());
        float hi = *max_element(im.begin(), im.end());
        Image<byte> out(im.size());
        for(int r=0; r < out.size().y-0; r++)
                for(int c=0; c < out.size().x-0; c++)
                        out[r][c] = (int)floor((im[r][c]-lo)*255/(hi-lo));

        return out;
}
```

8.2.2.6 Image<float> average_image (const vector< Image< float > > & *ims*)

Average the input image stack for display purposes.

Definition at line 151 of file multispot5.cc.

References assert_same_size().

```
{
    assert_same_size(ims);
    Image<float> r(ims[0].size(), 0);

    for(unsigned int i=0; i < ims.size(); i++)
        transform(r.begin(), r.end(), ims[i].begin(), r.begin(), plus<float>());

    transform(r.begin(), r.end(), r.begin(), bind2nd(multiplies<float>(), 1./ims.
      size()));
    return r;
}
```

8.2.2.7 vector<string> split (const string & *line*)

Tokenize a line.

Definition at line 915 of file multispot5.cc.

Referenced by parse_log_file().

```
{
    vector<string> v;
    istringstream i(line);
    string s;

    while(!i.eof())
    {
        i >> s;
        if(i.fail())
            break;
        v.push_back(s);
    }
    return v;
}
```

8.2.2.8 template<class C > string xtoa (const C & *x*) [inline]

Generic version of itoa.

How many times has this been reimplemented??

Parameters

x	Value to convert to string

Definition at line 935 of file multispot5.cc.

Referenced by parse_log_file().

```
{
    ostringstream os;
    os << x;
    return os.str();
}
```

8.2.2.9 template<class C > C atox (const string & *s*, const string & *msg*) [inline]

Inverse of xtoa() How many times has this been reimplemented??

Parameters

s	String to convert
msg	Mesage to print on error

Definition at line 947 of file multispot5.cc.

```
{
    C c;
    istringstream i(s);
    i >> c;
    if(i.fail())
        throw LogFileParseError("Error parsing " + msg + ". Text is '" + s + "'."
      );
    return c;
}
```

8.2.2.10 set<ImageRef> dilate_mask (const vector< ImageRef > & *v*, double *r*)

Very simple and inefficient dilation function.

Definition at line 1319 of file multispot5.cc.

```
{
    vector<ImageRef> m = getDisc(r);

    set<ImageRef> ret;

    for(unsigned int i=0; i < v.size(); i++)
        for(unsigned int j=0; j < m.size(); j++)
            ret.insert(v[i] + m[j]);

    return ret;
}
```

8.2.2.11 vector<int> SampledMultispot::sequence (int *n*) [inline]

Create a sequence of integers.

These can be used as observations in an observation class by forward_algorithm() and etc.

Parameters

n	Length of sequence

Definition at line 169 of file sampled_multispot.h.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes().

```
{
    vector<int> v;
    for(int i=0; i < n; i++)
        v.push_back(i);
    return v;
}
```

8.2.2.12 double sign (double *x*) [inline]

computes the sign of x

Parameters

x	*x*

Returns

$$\begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Definition at line 19 of file utility.h.

```
{
    return x>=0?1:-1;
}
```

8.2.2.13 float sq (float *f*) [inline]

The ubiquitous square function.

Parameters

f	Number to square

Returns

square of the number

Definition at line 29 of file utility.h.

Referenced by auto_fixed_scaling(), NegativeFreeEnergy::compute_with_mask(), diff_log_log_normal(), hess_log_log_normal(), log_log_normal(), log_normal_std(), log_probability_spot(), log_probability_spot_diff(), log_probability_spot_hess(), NegativeFreeEnergy::operator()(), spot_shape_diff_position(), spot_shape_hess_position(), and NegativeFreeEnergy::variance_from_sample().

```
{ return f*f; }
```

8.2.2.14 const std::vector<CVD::SubImage<float> > sub_images (const std::vector< CVD::Image< float > > & *im*, CVD::ImageRef *pos*, CVD::ImageRef *size*)

Cut sub images out of every member of a vector of images.

Parameters

im	Images to be cut
pos	Top left corner
size	Size of the patch

Returns

subimages.

Referenced by test_output_patch_variance().

8.2.2.15 std::pair<CVD::ImageRef, CVD::ImageRef> boundingbox (const std::vector< CVD::ImageRef > & *all_spots*)

Compute the bounding box of a set of points.

Parameters

all_spots	List of points

Referenced by mmain(), FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_modifying_model().

8.2.2.16 template<class Stream > void open_or_die (Stream & *save_spots,* const std::string & *save_spots_file*)

Parameters

save_spots	Stream
save_spots_file	File to open

Definition at line 64 of file utility.h.

Referenced by mmain().

```
{
    using std::cerr;
    using std::endl;
    using std::strerror;
    using std::exit;
    save_spots.open(save_spots_file.c_str());
    int err = errno;

    if(!save_spots.good())
    {
        cerr << "*********************************************************\n";
        cerr << "ERROR: failed to open " << save_spots_file << ": " <<strerror(er
      r) << endl;
        cerr << "*********************************************************\n";
        exit(1);
    }
}
```

8.3 Generic hidden Markov model solver.

The notation follows 'A tutorial on hidden Markov models and selected applications in speech recognition', Rabiner, 1989.

Functions

- template<int States, class Btype , class Otype >
  std::tr1::tuple< double, TooN::Vector< Btype::NumParameters >, TooN::Matrix< Btype::NumParameters > > forward_algorithm_hessian (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O, bool compute_deriv=1, bool compute_hessian=1)
- template<int States, class Btype , class Otype >
  double forward_algorithm (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)
- template<int States, class Btype , class Otype >
  std::pair< double, TooN::Vector< Btype::NumParameters > > forward_algorithm_deriv (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)

- template<int States, class Btype , class Otype >
  std::vector< std::tr1::array< double, States > > forward_algorithm_delta (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)
- template<int States, class Btype , class Otype >
  void forward_algorithm_delta2 (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O, std::vector< std::tr1::array< double, States > > &delta)
- template<int States, class Btype , class Otype >
  std::pair< std::vector< std::tr1::array< double, States > >, std::vector< std::tr1::array< double, States > > > forward_backward_algorithm (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)
- template<class A , class Rng >
  int select_random_element (const A &v, const double scale, Rng &rng)
- template<int N, class Rng >
  int sample_unscaled_log (std::tr1::array< double, N > a, Rng &rng)
- template<int States, class StateType , class Rng >
  std::vector< StateType > backward_sampling (TooN::Matrix< States > A, const std::vector< std::tr1::array< double, States > > &delta, Rng &rng)

8.3.1 Detailed Description

The notation follows 'A tutorial on hidden Markov models and selected applications in speech recognition', Rabiner, 1989.

8.3.2 Function Documentation

8.3.2.1 template<int States, class Btype , class Otype > std::tr1::tuple<double, TooN::Vector<Btype::NumParameters>, TooN::Matrix<Btype::NumParameters> > forward_algorithm_hessian (TooN::Matrix< States > *A*, TooN::Vector< States > *pi*, const Btype & *B*, const std::vector< Otype > & *O*, bool *compute_deriv* = 1, bool *compute_hessian* = 1)

The forward algorithm is defined as:

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{1}$$

$$\alpha_t(j) = b_j(O(t)) \sum_i \alpha_{t-1}(i) a_{ij} \tag{2}$$

And the probability of observing the data is just:

$$P(O_1 \cdots O_T|\lambda) = P(O|\lambda) \sum_i \alpha_T(i), \tag{3}$$

where the state, $\lambda = \{A, \pi, B\}$.

All multipliers are much less than 1, to α rapidly ends up as zero. Instead, store the logarithm:

$$\delta_t(i) = \ln \alpha_t(i) \tag{4}$$
$$\delta_1(i) = \ln \pi_i + \ln b_j(O_t) \tag{5}$$

and the recursion is:

$$\delta_t(j) = \ln b_j(O_t) + \ln \sum_i \alpha_{t-1}(i) a_{ij} \tag{6}$$
$$= \ln b_j(O_t) + \ln \sum_i e^{\delta_{t-1}(i) + \ln a_{ij}} \tag{7}$$
$$\tag{8}$$

including an arbitrary constant, $Z_t(j)$ gives:

$$\delta_t(j) = \ln b_j(O_t) + \ln \sum_i e^{Z_t(j)} e^{\delta_{t-1}(i) + \ln a_{ij} - Z_t(j)} \tag{9}$$
$$= \ln b_j(O_t) + Z_t(j) + \ln \sum_i e^{\delta_{t-1}(i) + \ln a_{ij} - Z_t(j)}. \tag{10}$$

In order to prevent a loss of scale on the addition:

$$Z_t(j) \stackrel{\text{def}}{=} \max_i \delta_{t-1}(i) + \ln a_{ij}, \tag{11}$$

so the largest exponent will be exactly 0. The final log probability is, similarly:

$$\ln P(O|\lambda) = Z + \ln \sum_i e^{\delta_T(i) - Z}, \tag{12}$$

Z can take any value, but to keep the numbers within a convenient range:

$$Z \stackrel{\text{def}}{=} \max_i \delta_T(i). \tag{13}$$

For computing derivatives, two useful results are:

$$\frac{\partial}{\partial x} \ln f(x) = \frac{f'(x)}{f(x)} \tag{14}$$
$$\frac{\partial}{\partial x} e^{f(x)} = f'(x) e^{f(x)} \tag{15}$$

There are M parameters of B, denoted $\phi_1 \ldots \phi_M$. The derivatives of P are:

$$\frac{\partial}{\partial \phi_n} P(O|\lambda) = \sum_{i-1}^{N} \frac{\partial}{\partial \phi_n} e^{\delta_T(i)} \tag{16}$$
$$= \sum_{i-1}^{N} e^{\delta_T(i)} \frac{\partial}{\partial \phi_n} \delta_T(i) \tag{17}$$
$$\tag{18}$$

Taking derivatives of $\ln P$ and rearranging to get numerically more convenient results gives:

$$\frac{\partial}{\partial \phi_n} \ln P(O|\lambda) = \frac{\frac{\partial}{\partial \phi_n} P(O|\lambda)}{P(O|\lambda)} \tag{19}$$

$$= \frac{\sum_{i-1}^{N} e^{\delta_T(i)} \frac{\partial}{\partial \phi_n} \delta_T(i)}{P(O|\lambda)} \tag{20}$$

$$= \sum_{i-1}^{N} e^{\delta_T(i) - \ln P(O|\lambda)} \frac{\partial}{\partial \phi_n} \delta_T(i) \tag{21}$$

The derivarives of δ are:

$$\frac{\partial \delta_T(j)}{\partial \phi_n} = \frac{\partial}{\partial \phi_n} \ln \left[b_j(O_t) \sum_{i-1}^{N} e^{\delta_{t-1}(i) + \ln a_{ij}} \right] \tag{22}$$

$$= \frac{\partial}{\partial \phi_n} [\ln b_j(O_t)] + \frac{\sum_{i-1}^{N} \frac{\partial}{\partial \phi_n} e^{\delta_{t-1}(i) + \ln a_{ij}}}{\sum_{i-1}^{N} e^{\delta_{t-1}(i) + \ln a_{ij}}} \tag{23}$$

$$\underbrace{\frac{\partial \delta_T(j)}{\partial \phi_n}}_{\texttt{diff\_delta[t][j]}} = \underbrace{\frac{\partial}{\partial \phi_n} [\ln b_j(O_t)]}_{\texttt{B.diff\_log(j, O[t])}} + \frac{\overbrace{\sum_{i-1}^{N} e^{\delta_{t-1}(i) + \ln a_{ij} - Z_t(j)} \frac{\partial}{\partial \phi_n} \delta_{t-1}(i)}^{\texttt{sum\_top}}}{\underbrace{\sum_{i-1}^{N} e^{\delta_{t-1}(i) + \ln a_{ij} - Z_t(j)}}_{\texttt{sum}}}, \tag{24}$$

with $Z_t(j)$ as defined in forward_algorithm.

For computing second derivatives, with ∇ yielding column vectors, two useful results are:

$$\mathscr{H} \ln f(\mathbf{x}) = \frac{\mathscr{H} f(\mathbf{x})}{f(\mathbf{x})} - \nabla f(\mathbf{x}) \nabla \mathbf{f}(\mathbf{x})^{\mathsf{T}} \tag{25}$$

$$\mathscr{H} e^{f}(\mathbf{x}) = e^{f(\mathbf{x})} (\nabla f(\mathbf{x}) \nabla \mathbf{f}(\mathbf{x})^{\mathsf{T}} + \mathscr{H} \mathbf{f}(\mathbf{x})), \tag{26}$$

therefore:

$$\mathscr{H} \ln P(O|\lambda) = \frac{\mathscr{H} f(\mathbf{x})}{P(O|\lambda)} - \nabla P(O|\lambda) \nabla P(O|\lambda)^{\mathsf{T}}, \tag{27}$$

and:

$$\mathscr{H} P(O|\lambda) = \sum_{i} e^{\delta_t(i) - \ln P(O|\lambda)} \left[\nabla \delta_t \nabla \delta_t^{\mathsf{T}} + \mathscr{H} \delta_t \right]. \tag{28}$$

Define $s_t(j)$ as:

$$s_t(j) = \sum_{i} e^{\delta_{t-1}(j) + \ln a_{ij}} \tag{29}$$

so that:

$$\delta_t(j) = \ln b_j(O_t) + \ln s_t(j). \tag{30}$$

The derivatives and Hessian recursion are therefore:

$$\nabla \delta_t(j) = \nabla \ln b_j(O_t) + \frac{\nabla s_t(j)}{s_t(j)} \tag{31}$$

$$\mathscr{H} \delta_t(j) = \mathscr{H} \ln b_j(O_t) + \frac{\mathscr{H} s_t(j)}{s_t(j)} - \frac{\nabla s_t(j)}{s_t(j)} \frac{\nabla s_t(j)}{s_t(j)}^{\mathsf{T}}. \tag{32}$$

$$= \underbrace{\mathscr{H} \ln b_j(O_t)}_{\texttt{B.hess\_log(j, O[t])}} + \frac{\overbrace{\sum_i e^{\delta_{t-1}(j) + \ln a_{ij} - Z_t(j)} \left[\mathscr{H} \delta_{t-1}(i) + \nabla \delta_{t-1}(i) \nabla \delta_{t-1}(i)^{\mathsf{T}} \right]}^{\texttt{sum\_top2}}}{\texttt{sum}} - \frac{\texttt{sum\_top sum\_top}^{\mathsf{T}}}{\texttt{sum}^2} \tag{33}$$

Parameters

A	*A:* State transition probabilities.
pi	π: initial state probabilities.
O	*O* or *I:* the observed data (ie the images).
B	*B*: A function object giving the (log) probability of an observation given a state, and derivatives with respect to the parameters.
compute_deriv	Whether to compute the derivative, or return zero.
compute_hessian	Whether to compute the Hessian, or return zero. This implies `compute_deriv`.

Returns

the log probability of observing all the data, and the derivatives of the log probability with respect to the parameters, and the Hessian.

Definition at line 134 of file forward_algorithm.h.

References ln().

Referenced by forward_algorithm_deriv(), sampled_background_spot_hessian2(), and sampled_background_spot_hessian_FAKE().

```
{
    using namespace TooN;
    using namespace std;
    using namespace std::tr1;

    if(compute_hessian == 1)
        compute_deriv=1;

    static const int M = Btype::NumParameters;
    int states = pi.size();

    //delta[j][i] = delta_t(i)
    vector<array<double, States> > delta(O.size());

    //diff_delta[t][j][n] = d/de_n delta_t(j)
    vector<array<Vector<M>,States > > diff_delta(O.size());
```

```
//hess_delta[t][j][m][n] = d2/de_n de_m delta_t(j)
vector<array<Matrix<M>,States > > hess_delta(O.size());

//Initialization: Eqn 19, P 262
//Set initial partial log probabilities:
for(int i=0; i < states; i++)
{
    delta[0][i] = ln(pi[i]) + B.log(i, O[0]);

    if(compute_deriv)
        diff_delta[0][i] = B.diff_log(i, O[0]);

    if(compute_hessian)
        hess_delta[0][i] = B.hess_log(i, O[0]);
}

//Perform the recursion: Eqn 20, P262
//Note, use T and T-1. Rather than  T+1 and T.
for(unsigned int t=1; t < O.size(); t++)
{
    for(int j=0; j < states; j++)
    {
        double Ztj = -HUGE_VAL; //This is Z_t(j)
        for(int i=0; i < states; i++)
            Ztj = max(Ztj, delta[t-1][i] + ln(A[i][j]));

        double sum=0;
        for(int i=0; i < states; i++)
            sum += exp(delta[t-1][i] + ln(A[i][j]) - Ztj);

        delta[t][j] = B.log(j, O[t]) + Ztj + ln(sum);

        if(compute_deriv)
        {
            Vector<M> sum_top = Zeros;
            for(int i=0; i < states; i++)
                sum_top += diff_delta[t-1][i] * exp(delta[t-1][i] + ln(A[i][j
  ]) - Ztj);

            diff_delta[t][j] = B.diff_log(j, O[t]) +  (sum_top) / sum;

            if(compute_hessian)
            {
                Matrix<M> sum_top2 = Zeros;
                for(int i=0; i < states; i++)
                    sum_top2 += exp(delta[t-1][i] + ln(A[i][j]) - Ztj) * ( he
  ss_delta[t-1][i] + diff_delta[t-1][i].as_col() * diff_delta[t-1][i].as_row());

                hess_delta[t][j] = B.hess_log(j, O[t]) + sum_top2 / sum - sum
  _top.as_col() * sum_top.as_row() / (sum*sum);
            }
        }
    }
}

//Compute the log prob using normalization
double Z = -HUGE_VAL;
for(int i=0; i < states; i++)
    Z = max(Z, delta.back()[i]);

double sum =0;
for(int i=0; i < states; i++)
```

```
        sum += exp(delta.back()[i] - Z);

    double log_prob = Z  + ln(sum);

    //Compute the differential of the log
    Vector<M> diff_log = Zeros;
    //Compute the differential of the log using normalization
    //The convenient normalizer is ln P(O|lambda) which makes the bottom 1.
    for(int i=0; compute_deriv && i < states; i++)
        diff_log += exp(delta.back()[i] - log_prob)*diff_delta.back()[i];

    Matrix<M> hess_log = Zeros;
    //Compute the hessian of the log using normalization
    //The convenient normalizer is ln P(O|lambda) which makes the bottom 1.
    for(int i=0; compute_hessian && i < states; i++)
        hess_log += exp(delta.back()[i] - log_prob) * (hess_delta.back()[i] + dif
      f_delta.back()[i].as_col() * diff_delta.back()[i].as_row());

    hess_log -= diff_log.as_col() * diff_log.as_row();

    //Compute the differential of the Hessian
    return  make_tuple(log_prob, diff_log, hess_log);
}
```

8.3.2.2 template<int States, class Btype , class Otype > double forward_algorithm (TooN::Matrix< States > *A*, TooN::Vector< States > *pi*, const Btype & *B*, const std::vector< Otype > & *O*)

Run the forward algorithm and return the log probability.

Parameters

A	*A:* State transition probabilities.
pi	π: initial state probabilities.
O	*O* or *I:* the observed data (ie the images).
B	*B*: A function object giving the (log) probability of an observation given a state.

Returns

the log probability of observing all the data.

Definition at line 244 of file forward_algorithm.h.

References ln().

```
{
    using namespace TooN;
    using namespace std;
    using namespace std::tr1;

    int states = pi.size();

    //delta[j][i] = delta_t(i)
    vector<array<double, States> > delta(O.size());
```

```
    //Initialization: Eqn 19, P 262
    //Set initial partial log probabilities:
    for(int i=0; i < states; i++)
        delta[0][i] = ln(pi[i]) + B.log(i, O[0]);

    //Perform the recursion: Eqn 20, P262
    //Note, use T and T-1. Rather than  T+1 and T.
    for(unsigned int t=1; t < O.size(); t++)
    {
        for(int j=0; j < states; j++)
        {
            double Ztj = -HUGE_VAL; //This is Z_t(j)
            for(int i=0; i < states; i++)
                Ztj = max(Ztj, delta[t-1][i] + ln(A[i][j]));

            double sum=0;
            for(int i=0; i < states; i++)
                sum += exp(delta[t-1][i] + ln(A[i][j]) - Ztj);

            delta[t][j] = B.log(j, O[t]) + Ztj + ln(sum);
        }
    }

    //Compute the log prob using normalization
    double Z = -HUGE_VAL;
    for(int i=0; i < states; i++)
        Z = max(Z, delta.back()[i]);

    double sum =0;
    for(int i=0; i < states; i++)
        sum += exp(delta.back()[i] - Z);

    double log_prob = Z  + ln(sum);

    return  log_prob;
}
```

8.3.2.3 template<int States, class Btype , class Otype > std::pair<double, TooN::Vector<Btype::NumParameters> > forward_algorithm_deriv (TooN::Matrix< States > *A*, TooN::Vector< States > *pi*, const Btype & *B*, const std::vector< Otype > & *O*)

Run the forward algorithm and return the log probability and its derivatives.

Parameters

A	*A:* State transition probabilities.
pi	π: initial state probabilities.
O	*O* or *I:* the observed data (ie the images).
B	*B*: A function object giving the (log) probability of an observation given a state, and derivatives with respect to the parameters.

Returns

the log probability of observing all the data.

Definition at line 302 of file forward_algorithm.h.

References forward_algorithm_hessian().

Referenced by SpotNegProbabilityDiffWithSampledBackground::operator()().

```
{
    using namespace std::tr1;
    double p;
    TooN::Vector<Btype::NumParameters> v;
    tie(p,v, ignore) = forward_algorithm_hessian(A, pi, B, O, 1, 0);
    return make_pair(p,v);
}
```

8.3.2.4 template<int States, class Btype , class Otype > std::vector<std::tr1::array<double, States>> forward_algorithm_delta (TooN::Matrix< States > *A*, TooN::Vector< States > *pi*, const Btype & *B*, const std::vector< Otype > & *O*)

Run the forward algorithm and return the log partials (delta)

Parameters

A	*A:* State transition probabilities.
pi	π: initial state probabilities.
O	*O* or *I:* the observed data (ie the images).
B	B: A function object giving the (log) probability of an observation given a state, and derivatives with respect to the parameters.

Returns

the log probability of observing all the data.

Definition at line 323 of file forward_algorithm.h.

References ln().

Referenced by forward_backward_algorithm(), SampledMultispot::GibbsSampler::next(), and sampled_background_spot_hessian_ffbs().

```
{
    using namespace TooN;
    using namespace std;
    using namespace std::tr1;

    int states = pi.size();

    //delta[j][i] = delta_t(i)
    vector<array<double, States> > delta(O.size());

    //Initialization: Eqn 19, P 262
    //Set initial partial log probabilities:
    for(int i=0; i < states; i++)
        delta[0][i] = ln(pi[i]) + B.log(i, O[0]);

    //Forward pass...
```

```
    //Perform the recursion: Eqn 20, P262
    //Note, use T and T-1. Rather than  T+1 and T.
    for(unsigned int t=1; t < O.size(); t++)
    {
        for(int j=0; j < states; j++)
        {
            double Ztj = -HUGE_VAL; //This is Z_t(j)
            for(int i=0; i < states; i++)
                Ztj = max(Ztj, delta[t-1][i] + ln(A[i][j]));

            double sum=0;
            for(int i=0; i < states; i++)
                sum += exp(delta[t-1][i] + ln(A[i][j]) - Ztj);

            delta[t][j] = B.log(j, O[t]) + Ztj + ln(sum);
        }
    }

    return delta;
}
```

8.3.2.5 template<int States, class Btype , class Otype > void forward_algorithm_delta2 (TooN::Matrix< States > *A,* TooN::Vector< States > *pi,* const Btype & *B,* const std::vector< Otype > & *O,* std::vector< std::tr1::array< double, States > > & *delta*)

Run the forward algorithm and return the log partials (delta)

Parameters

A	*A:* State transition probabilities.
pi	π: initial state probabilities.
O	*O* or *I:* the observed data (ie the images).
B	B: A function object giving the (log) probability of an observation given a state, and derivatives with respect to the parameters.
delta	the δ values

Definition at line 371 of file forward_algorithm.h.

References ln().

```
{
    using namespace TooN;
    using namespace std;
    using namespace std::tr1;

    int states = pi.size();

    //delta[j][i] = delta_t(i)
    delta.resize(O.size());

    //Initialization: Eqn 19, P 262
    //Set initial partial log probabilities:
    for(int i=0; i < states; i++)
        delta[0][i] = ln(pi[i]) + B.log(i, O[0]);
```

```
    Matrix<States> lA;
    for(int r=0; r < States; r++)
        for(int c=0; c < States; c++)
            lA[r][c] = ln(A[r][c]);

    //Forward pass...
    //Perform the recursion: Eqn 20, P262
    //Note, use T and T-1. Rather than  T+1 and T.
    for(unsigned int t=1; t < O.size(); t++)
    {
        for(int j=0; j < states; j++)
        {
            double Ztj = -HUGE_VAL; //This is Z_t(j)
            for(int i=0; i < states; i++)
                Ztj = max(Ztj, delta[t-1][i] + lA[i][j]);

            double sum=0;
            for(int i=0; i < states; i++)
                sum += exp(delta[t-1][i] + lA[i][j] - Ztj);

            delta[t][j] = B.log(j, O[t]) + Ztj + ln(sum);
        }
    }
}
```

8.3.2.6 template<int States, class Btype , class Otype > std::pair<std::vector<std::tr1::array<double, States> >, std::vector<std::tr1::array<double, States> > > forward_backward_algorithm (TooN::Matrix< States > *A*, TooN::Vector< States > *pi*, const Btype & *B*, const std::vector< Otype > & *O*)

Run the forward-backwards algorithm and return the log partials (delta and epsilon).

Parameters

A	*A:* State transition probabilities.
pi	π: initial state probabilities.
O	*O* or *I:* the observed data (ie the images).
B	B: A function object giving the (log) probability of an observation given a state, and derivatives with respect to the parameters.

Returns

the log probability of observing all the data.

Backward pass Epsilon is log beta

Definition at line 424 of file forward_algorithm.h.

References forward_algorithm_delta(), and ln().

```
{
    using namespace TooN;
    using namespace std;
    using namespace std::tr1;
```

```
    int states = pi.size();

    //delta[j][i] = delta_t(i)
    vector<array<double, States> > delta = forward_algorithm_delta(A, pi, B, O);

    ///Backward pass
    ///Epsilon is log beta
    vector<array<double, States> > epsilon(O.size());

    //Initialize beta to 1, ie epsilon to 0
    for(int i=0; i < states; i++)
        epsilon[O.size()-1][i] = 0;

    //Perform the backwards recursion
    for(int t=O.size()-2; t >= 0; t--)
    {
        for(int i=0; i < states; i++)
        {
            //Find a normalizing constant
            double Z = -HUGE_VAL;

            for(int j=0; j < states; j++)
                Z = max(Z, ln(A[i][j]) + B.log(j, O[t+1]) + epsilon[t+1][j]);

            double sum=0;
            for(int j= 0; j < states; j++)
                sum += exp(ln(A[i][j]) + B.log(j, O[t+1]) + epsilon[t+1][j] - Z);


            epsilon[t][i] = ln(sum) + Z;
        }
    }

    return make_pair(delta, epsilon);
}
```

8.3.2.7 template<class A , class Rng > int select_random_element (const A & *v,* const double *scale,* Rng & *rng*)

Select an element from the container v, assuming that v is a probability distribution over elements up to some scale.

Parameters

v	Uscaled probability distribution
scale	Scale of v
rng	Random number generator to use

Definition at line 479 of file forward_algorithm.h.

References scale().

Referenced by sample_unscaled_log().

```
{
```

```
    double total=0, choice = rng()*scale;

    for(int i=0; i < (int)v.size(); i++)
    {
        total += v[i];
        if(choice <= total)
            return i;
    }
    return v.size()-1;
}
```

8.3.2.8 template<int N, class Rng > int sample_unscaled_log (std::tr1::array< double, N > *a,* Rng & *rng*)

Select an element from the a, assuming that a stores unscaled log probabilities of the elements.

Parameters

a	Uscaled probability distribution, stored as logarithms.
rng	Random number generator to use

Definition at line 497 of file forward_algorithm.h.

References select_random_element().

```
{
    double hi = *max_element(a.begin(), a.end());
    double sum=0;

    for(unsigned int i=0; i < a.size(); i++)
    {
        a[i] = exp(a[i] - hi);
        sum += a[i];
    }

    return select_random_element(a, sum, rng);
}
```

8.3.2.9 template<int States, class StateType , class Rng > std::vector<StateType> backward_sampling (TooN::Matrix< States > *A,* const std::vector< std::tr1::array< double, States > > & *delta,* Rng & *rng*)

An implementation of the backwards sampling part of the forwards filtering/backwards sampling algorithm.

See 'Monte Carlo smoothing for non-linear time series', Godsill and Doucet, JASA 2004

Parameters

A	HMM transition matrix.
delta	Forward partial probabilities stored as logarithms.
rng	Random number generator to use

Returns

state at each time step.

Definition at line 519 of file forward_algorithm.h.

References ln().

```
{
    //Compute the elementwise log of A
    for(int r=0; r < A.num_rows(); r++)
        for(int c=0; c < A.num_cols(); c++)
            A[r][c] = ln(A[r][c]);

    std::vector<StateType> samples(delta.size());

    samples.back() = sample_unscaled_log<States, Rng>(delta.back(), rng);

    //A is A[t][t+1]

    for(int i=delta.size()-2; i >= 0; i--)
    {
        std::tr1::array<double, States> reverse_probabilities = delta[i];

        for(int j=0; j < States; j++)
            reverse_probabilities[j] += A[j][samples[i+1]];

        samples[i] = sample_unscaled_log<States, Rng>(reverse_probabilities, rng)
      ;
    }

    return samples;
}
```

8.4 Classes related to the ImageJ Plugin

Classes

- class ClassicGlow

  Plugin implementing the classic glow/hot LUT which seems to be missing from ImageJ.
- class LoadTestData

  Plugin class to load the standard test data from the JAR file.
- class JNIUserInterface

  3B User interface for the Java plugin.
- class SPair

  Utility class to hold a pair of strings.
- class Util

  Utility calss to hold a number of handy static functions.
- class three_B

  ImageJ plugin class.
- class AdvancedDialog

Control panel which basically presents the .cfg file in a large edit box along with additional necessary things (frame range and pixel size).

- class ThreeBDialog

  Dialog box for starting 3B The dialog highlights bad things in red.
- class Spot

  Basic spot class, simply contains coordinates.
- class SomethingChanges

  Listener class which triggers a complete redraw.
- class FloatSliderWithBox

  This class makes a floating point slider bar with an edit box next to it for more precision.
- class CloseButtonListener

  Close button issues a window close event.
- class StopButtonListener

  Stop 3B thread.
- class ExportButtonListener

  Listener for the export button.
- class EControlPanel

  Control panel for running 3B plugin and providing interactive update.
- class ThreeBRunner

  This class deals with running the actual 3B code It should be run in a separate thread It will make calls back to a viewer, and accepts termination calls as well.
- class ThreeBHelp

  3B plugin cclass to bring up a basic help window.
- class ThreeBLoader

  Plugin class to load up an old 3B run.

Functions

- string get_string (JNIEnv ∗env, jstring js)
- Image< jbyte > get_local_copy_of_image (JNIEnv ∗env, jbyteArray data, int rows, int cols)
- Image< float > get_local_copy_of_image (JNIEnv ∗env, jfloatArray data, int rows, int cols)
- JNIEXPORT void JNICALL Java_ThreeBRunner_call (JNIEnv ∗env, jobject jthis, jstring cfg, jobjectArray images, jbyteArray mask_data, jint n_images, jint rows, jint cols, jstring file)
- static SPair Util::getFileName (String t)

8.4.1 Function Documentation

8.4.1.1 string get_string (JNIEnv * *env*, jstring *js*)

Get a local C++ copy of a java string.

Definition at line 122 of file multispot5_jni.cc.

Referenced by Java_ThreeBRunner_call().

```
{
    const char* str;

    //Covert the config into a string
    str = env->GetStringUTFChars(js, NULL);

    string stdstring(str);
    env->ReleaseStringUTFChars(js, str);


    return stdstring;
}
```

8.4.1.2 Image<jbyte> get_local_copy_of_image (JNIEnv * *env*, jbyteArray *data*, int *rows*, int *cols*)

Get a local C++ copy of an image from a jbyteArray coming from the guts of ImageJ.

Definition at line 138 of file multispot5_jni.cc.

Referenced by Java_ThreeBRunner_call().

```
{
    //This takes a copy of the pixels (perhaps)
    jbyte* pix = env->GetByteArrayElements(data, NULL);

    BasicImage<jbyte> pix_im(pix, ImageRef(cols, rows));

    Image<jbyte> im;
    im.copy_from(pix_im);

    //This frees the pixels if copied, or releases a reference
    env->ReleaseByteArrayElements(data,pix, JNI_ABORT);

    return im;
}
```

8.4.1.3 Image<float> get_local_copy_of_image (JNIEnv * *env*, jfloatArray *data*, int *rows*, int *cols*)

Get a local C++ copy of an image from a jfloatArray coming from the guts of ImageJ.

Definition at line 156 of file multispot5_jni.cc.

```
{
    //This takes a copy of the pixels (perhaps)
```

```
    float* pix = env->GetFloatArrayElements(data, NULL);

    BasicImage<float> pix_im(pix, ImageRef(cols, rows));

    Image<float> im;
    im.copy_from(pix_im);

    //This frees the pixels if copied, or releases a reference
    env->ReleaseFloatArrayElements(data,pix, JNI_ABORT);

    return im;
}
```

8.4.1.4 JNIEXPORT void JNICALL Java_ThreeBRunner_call (JNIEnv * *env*, jobject *jthis*, jstring *cfg*, jobjectArray *images*, jbyteArray *mask_data*, jint *n_images*, jint *rows*, jint *cols*, jstring *file*)

Run the 3B code.

Definition at line 176 of file multispot5_jni.cc.

References JNIUserInterface::fatal(), get_local_copy_of_image(), get_string(), null_graphics(), and JNIUserInterface::send_message().

```
{
    istringstream config(get_string(env, cfg));
    GUI.ParseStream(config);

    JNIUserInterface ui(env, jthis);
    ui.send_message("Initializing...");

    string filename = get_string(env, file);

    //Attmpt to open the file
    ofstream save_spots;
    save_spots.open(filename.c_str());
    int err = errno;

    if(!save_spots.good())
    {
        ui.fatal("failed to open " + filename + ": " + strerror(err));
        return;
    }

    vector<ImageRef> maskir;
    Image<double> maskd;
    {
        Image<jbyte> mask = get_local_copy_of_image(env, mask_data, rows, cols);
        maskd = convert_image(mask);
        for(ImageRef p(-1, 0); p.next(mask.size()); )
            if(mask[p])
                maskir.push_back(p);
    }


    vector<Image<float> > ims;

    for(int i=0; i < n_images; i++)
```

```
    {
        jfloatArray f = static_cast<jfloatArray>(env->GetObjectArrayElement(image
      s, i));
        ims.push_back(preprocess_image(get_local_copy_of_image(env, f, rows, cols
      )));
        env->DeleteLocalRef(f);
    }

    double mean, variance;
    tie(mean, variance) = mean_and_variance(ims);

    for(unsigned int i=0; i < ims.size(); i++)
            transform(ims[i].begin(), ims[i].end(), ims[i].begin(), bind1st(multi
      plies<double>(), 1/ sqrt(variance)));

    tie(mean, variance) = mean_and_variance(ims);

    //A sanity check.
    cerr << "Rescaled:\n";
    cerr << "mean = " << mean << endl;
    cerr << "std  = " << sqrt(variance) << endl;
    cerr << "Version 1.1" << endl;


    auto_ptr<FitSpotsGraphics> gr = null_graphics();

    place_and_fit_spots(ims, maskir, maskd, save_spots, *gr, ui);
}
```

8.4.1.5 static SPair Util::getFileName (String *t*) [inline, static, package]

The a file name for saving and complete path using ImageJ's file open dialog.

Kep re-querying user if the file will be overwritten. Windows already provides the query built in.

Parameters

t	Initial file name

Returns

filename and full path

Definition at line 80 of file three_B.java.

References SPair::a, and SPair::b.

Referenced by three_B::run().

```
    {
        //Get a filename to save as, with appropriate warnings for
        //overwriting files.
        String fname, fullname;
        while(true)
        {
            SaveDialog save = new SaveDialog("Save 3B output", t, ".txt");
```

```
            fname = save.getFileName();

            fullname = save.getDirectory() + File.separator + fname;

            if(fname == null)
                break;

            File test = new File(fullname);
            //Windows' open dialog seems to do overwrite confirmation automatical
    ly,
            //so there is no need to do it here.
            if(!ij.IJ.isWindows() && test.exists())
            {
                GenericDialog g = new GenericDialog("Overwrite file?");
                g.addMessage("The file \"" + fname + "\" already exists. Continue
     and overwrite?");
                g.enableYesNoCancel("Yes", "No");
                g.showDialog();

                if(g.wasOKed())
                    break;
                else if(g.wasCanceled())
                {
                    fname = null;
                    break;
                }
            }
            else
                break;
        }

        SPair r = new SPair();
        r.a = fname;
        r.b = fullname;
        return r;
    }
```

8.5 Storm classes

Classes

- struct ConjugateGradientOnly< Size, Precision >

  Class for performing optimization with Conjugate Gradient, where only the derivatives are available.
- class NullGraphics

  Graphics class which does absoloutely nothing.
- class DataForMCMC

  Closure hoding the data required do use GibbsSampler2 See FitSpots for naming of variables.
- struct SampledBackgroundData

  Closure holding image data generated using samples drawn from the model.
- struct SpotNegProbabilityDiffWithSampledBackground

Compute the derivative of the negative log probability with respect to the parameters of one spot, given some samples of the other spots.

- class FreeEnergyHessian

  Class for computing the Hessian of the negative free energy.
- class FitSpots

  Mega class which actually does the meat of the spot fitting.
- class FitSpotsGraphics

  Graphics class for FittingSpots.
- class UserInterfaceCallback

  Callback class used by FitSpots to provide enough hooks for a user interface.
- class SampledMultispot::GibbsSampler

  Draw samples from the spot states given the spots positions and some data.
- class SampledMultispot::GibbsSampler2

  Gibbs sampling class which masks spots to reduce computation.

Functions

- auto_ptr< UserInterfaceCallback > null_ui ()
- auto_ptr< FitSpotsGraphics > null_graphics ()
- void get_spot_pixels (const vector< ImageRef > &pixels, const Vector< 4 > &spot, vector< int > &out)
- StateParameters parse_log_file (istream &in)
- double brightness_motion_limit (double mu, double sigma, bool not_one)
- void fit_spots_new (const vector< Image< float > > &ims, StateParameters &p, ofstream &save_spots, FitSpotsGraphics &gr)
- void fit_spots_new (const vector< Image< float > > &ims, StateParameters &p, ofstream &save_spots, FitSpotsGraphics &gr, UserInterfaceCallback &ui)
- template<class B >
  double spot_shape_s (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::pair< double, TooN::Vector< 4 > > spot_shape_diff_position (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::tr1::tuple< double, TooN::Vector< 4 >, TooN::Matrix< 4 > > spot_shape_hess_position (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::tr1::tuple< double, TooN::Vector< 2 >, TooN::Matrix< 2 > > spot_shape_hess (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::pair< double, TooN::Vector< 2 > > spot_shape_diff (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  double spot_shape (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)

- template<class Base >
  std::tr1::tuple< double, TooN::Vector< 2 >, TooN::Matrix< 2 > > log_probability_spot_hess (const CVD::SubImage< float > &im, double variance, const TooN::Vector< 4, double, Base > &spot_parameters)
- template<class Base >
  std::pair< double, TooN::Vector< 2 > > log_probability_spot_diff (const CVD::SubImage< float > &im, double variance, const TooN::Vector< 4, double, Base > &spot_parameters)
- template<class Base >
  double log_probability_spot (const CVD::SubImage< float > &im, double variance, const TooN::Vector< 4, double, Base > &spot_parameters)
- double log_normal_std (double mu, double sigma)
- double log_normal_mode (double mu, double sigma)
- double log_log_normal (double x, double mu, double sigma)
- double diff_log_log_normal (double x, double mu, double sigma)
- double hess_log_log_normal (double x, double mu, double sigma)

8.5.1 Function Documentation

8.5.1.1 auto_ptr<UserInterfaceCallback> null_ui ( )

Factory function to generate an instance of NullGraphics.

Definition at line 62 of file multispot5.cc.

Referenced by fit_spots_new().

```
{
    return auto_ptr<UserInterfaceCallback>(new NullUICallback);
}
```

8.5.1.2 auto_ptr<FitSpotsGraphics> null_graphics ( )

Factory function to generate an instance of NullGraphics.

Definition at line 89 of file multispot5.cc.

Referenced by Java_ThreeBRunner_call(), and mmain().

```
{
    return auto_ptr<FitSpotsGraphics>(new NullGraphics);
}
```

8.5.1.3 void get_spot_pixels (const vector< ImageRef > & *pixels,* const Vector< 4 > & *spot,* vector< int > & *out*)

Which pixels belong to a given spot? Find the indices of those pixels.

Definition at line 886 of file multispot5.cc.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_-modifying_model().

```
{
    //Go out to three sigma

    vector<ImageRef> pix = getDisc(spot[1]*6 + 1);
    out.resize(0);
    ImageRef offset = ir_rounded(spot.slice<2,2>());
    for(unsigned int j=0; j < pix.size(); j++)
    {
        int pos = lower_bound(pixels.begin(), pixels.end(), pix[j] + offset) - pi
      xels.begin();
        if(pos != (int)pixels.size() && pixels[pos] == pix[j] + offset)
            out.push_back(pos);
    }

    if(out.size() == 0)
    {
        cout << "*******************************\n";
        cout << "*******************************\n";
        cout << "*******************************\n";
        cout << "*******************************\n";
        cout << "*******************************\n";
        cout << "Oe noes!11one\n";
        cout << pix.size() << endl;
    }
}
```

8.5.1.4 StateParameters parse_log_file (istream & *in*)

Parser for multispot 5 log files.

Log files are mostly line oriented and contain various records

The main records are:

Iteraton: #ITNUM MAIN: <list of spot parameters>

Pass: #PASSNUM MT19937 <random number generator state> PASS#PASSNUM: <list of spot parameters> ENDCHECKPOINT

Note that MAIN is redundant since it will be the same as the following PASS 1 (or the first pass computed if restoring from a checkpoint).

Data should only be considered valid after ENDCHECKPOINT has been read

Iteration is written once per iteration, not once per pass. (FIXME)

Which moron invented this file format???

Note that the file format hasn't beren fixed, so that the output can easily be compared

to the output of the historic version which is known to be good.

Parameters

in	Stream to parse file from

Definition at line 986 of file multispot5.cc.

References StateParameters::iteration, StateParameters::pass, StateParameters::pixels, StateParameters::rng, split(), StateParameters::spots, and xtoa().

Referenced by mmain().

```
{
    //A line read from the file
    string line;

    //State lines known to be OK
    string rngline, passline, iterationline;
    bool state_ok=0;

    //State lines read in, with flags of goodness
    string new_rngline, new_passline, new_iterationline;
    bool new_rngline_ok=0, new_passline_ok=0, new_iterationline_ok=0;

    unsigned int lineno=0;
    bool doing_gvars = 0;

    vector<ImageRef> pixels;

    while(!in.eof())
    {
        getline(in, line);
        if(in.fail())
            break;

        lineno++;

        if(line == "ENDGVARLIST")
        {
            if(!doing_gvars)
                throw LogFileParseError("Spurious end of GVars");
            doing_gvars = 0;
        }
        else if(doing_gvars)
        {
            GUI.ParseLine(line);
        }
        else if(line == "BEGINGVARLIST")
        {
            doing_gvars = 1;
        }
        if(line.substr(0, 11)  == "Iteration: ")
        {
            new_iterationline = line;
            new_iterationline_ok = true;
        }
        else if(line.substr(0, 4) == "PASS")
        {
            new_passline = line;
            if(new_passline_ok)
```

```
            throw LogFileParseError("Duplicate PASS on line " + xtoa(lineno))
    ;
            new_passline_ok = true;
        }
        else if(line.substr(0, 8) == "MT19937 ")
        {
            new_rngline = line;
            if(new_rngline_ok)
                throw LogFileParseError("Duplicate MT19937 on line " + xtoa(linen
    o));

            new_rngline_ok = true;

        }
        else if(line == "ENDCHECKPOINT")
        {
            if(new_passline_ok && new_rngline_ok && new_iterationline_ok)
            {
                iterationline = new_iterationline;
                rngline = new_rngline;
                passline = new_passline;
            }
            else
                throw LogFileParseError("Reached checkpoint with missing data: "
                         "it=" + xtoa(new_iterationline_ok) +
                        " pa=" + xtoa(new_passline_ok) +
                        " rg=" + xtoa(new_rngline_ok) + " on line " + xtoa(lineno
    ));

            //Don't reset iteration since it only appears once for each entire
            //set of passes.
            new_rngline_ok = 0;
            new_passline_ok = 0;

            state_ok = true;
        }
        else if(line.substr(0, 7) == "PIXELS ")
        {
            vector<string> l = split(line);
            if( (l.size() - 1)%2 == 0)
            {
                int n = (l.size()-1)/2;
                pixels.resize(n);
                for(int i=0; i < n; i++)
                {
                    pixels[i].x = atox<int>(l[i*2+1+0], "pixels");
                    pixels[i].y = atox<int>(l[i*2+1+1], "pixels");
                }
            }
            else
                throw LogFileParseError("Bad PIXELS line");
        }
    }

    if(!state_ok)
        throw LogFileParseError("No state found");

    if(pixels.size() == 0)
        throw LogFileParseError("No pixels, or pixels is empty");

    //Now parse the lines
    StateParameters p;
```

```
    vector<string> l;

    //Parse the iterations
    l = split(iterationline);
    p.iteration = atox<int>(l[1], "iteration");

    //Parse the random number generator
    p.rng =shared_ptr<MT19937>(new MT19937);
    {
        istringstream rng_s(rngline);
        try{
            p.rng->read(rng_s);
        }
        catch(MT19937::ParseError p)
        {
            throw LogFileParseError("Error parsing MT19937");
        }
    }

    //Parse PASS and the listing of spots
    l = split(passline);
    if( (l.size() - 1 ) % 4 == 0)
    {
        p.pass = atox<int>(l[0].substr(4), "pass");

        for(unsigned int i=0; i < (l.size()-1)/4; i++)
        {
            cout << l[i*4+1+0] << endl;
            cout << l[i*4+1+1] << endl;
            cout << l[i*4+1+2] << endl;
            cout << l[i*4+1+3] << endl;
            p.spots.push_back(makeVector(
                        atox<double>(l[i*4+1+0], "spot"),
                        atox<double>(l[i*4+1+1], "spot"),
                        atox<double>(l[i*4+1+2], "spot"),
                        atox<double>(l[i*4+1+3], "spot")));
        }

    }
    else
        throw LogFileParseError("Wrong number of elements in PASS line");

    //Set up the pixels (oh god the pixels)
    p.pixels = pixels;

    return p;
}
```

8.5.1.5 double brightness_motion_limit (double *mu,* double *sigma,* bool *not_one*)

How far should steps in brightness be limited to?

Definition at line 1334 of file multispot5.cc.

References log_normal_std().

```
{
    if(not_one)
        return log_normal_std(mu, sigma);
```

```
    else
        return 1;
}
```

8.5.1.6 void fit_spots_new (const vector< Image< float > > & *ims,* StateParameters & *p,* ofstream & *save_spots,* FitSpotsGraphics & *gr*)

Wrapper function for using FitSpots.

Definition at line 2081 of file multispot5.cc.

References null_ui(), and FitSpots::run().

Referenced by mmain().

```
{
    auto_ptr<UserInterfaceCallback> ui = null_ui();
    FitSpots fit(ims, gr, *ui, p, save_spots);
    fit.run();
}
```

8.5.1.7 void fit_spots_new (const vector< Image< float > > & *ims,* StateParameters & *p,* ofstream & *save_spots,* FitSpotsGraphics & *gr,* UserInterfaceCallback & *ui*)

Wrapper function for using FitSpots.

Definition at line 2090 of file multispot5.cc.

References FitSpots::run().

```
{
    try{
        FitSpots fit(ims, gr, ui, p, save_spots);
        fit.run();
    }
    catch(UserInterfaceCallback::UserIssuedStop)
    {
    }
}
```

8.5.1.8 template<class B > double spot_shape_s (const TooN::Vector< 2 > & *x,* const TooN::Vector< 4, double, B > & *phi*)

See spot_shape()

Parameters

x	$\mathbf{x}$
phi	ϕ

Returns

$s(\mathbf{x}, \phi)$

Definition at line 17 of file storm.h.

Referenced by spot_shape(), spot_shape_diff(), spot_shape_diff_position(), spot_shape_hess(), and spot_shape_hess_position().

```
{
    return -norm_sq(x - phi.template slice<2,2>()) / (2*phi[1]*phi[1]);
}
```

8.5.1.9 template<class B > std::pair<double, TooN::Vector<4> > spot_shape_diff_position (const TooN::Vector< 2 > & *x*, const TooN::Vector< 4, double, B > & *phi*)

Compute the spot shape and its derivative with respect to posision.

See also spot_shape()

Parameters

x	$\mathbf{x}$
phi	ϕ

Definition at line 27 of file storm.h.

References spot_shape_s(), and sq().

```
{
    using namespace TooN;

    double s = spot_shape_s(x, phi);
    double r_2_pi = sqrt(2*M_PI);

    double prob = exp(s) * phi[0]/(phi[1]*r_2_pi);

    Vector<4> deriv = (exp(s) / (phi[1]*r_2_pi)) *
                          makeVector(1,
                                     -phi[0] * (1 + 2*s)/phi[1],
                                     (x[0] - phi[2])*(phi[0]/sq(phi[1])),
                                     (x[1] - phi[3])*(phi[0]/sq(phi[1])));
    return std::make_pair(prob, deriv);
}
```

8.5.1.10 template<class B > std::tr1::tuple<double, TooN::Vector<4>, TooN::Matrix<4> > spot_shape_hess_position (const TooN::Vector< 2 > & *x*, const TooN::Vector< 4, double, B > & *phi*)

Compute the spot shape and its Hessian with respect to posision.

See also spot_shape()

Parameters

x	**x**
phi	ϕ

Definition at line 49 of file storm.h.

References spot_shape_s(), and sq().

```
{
    using namespace TooN;
    using namespace std::tr1;

    double s = spot_shape_s(x, phi);
    double r_2_pi = sqrt(2*M_PI);

    double es = exp(s);

    double prob = es * phi[0]/(phi[1]*r_2_pi);

    Vector<4> deriv = (es / (phi[1]*r_2_pi)) *
                          makeVector(1,
                                     -phi[0] * (1 + 2*s)/phi[1],
                                     (x[0] - phi[2])*(phi[0]/sq(phi[1])),
                                     (x[1] - phi[3])*(phi[0]/sq(phi[1])));

    Matrix<4> hess;
    hess[0][0] = 0;

    hess[0][1] = -es*(1+2*s) / (phi[1] * phi[1] * r_2_pi);
    hess[1][0] = hess[0][1];

    hess[0][2] = es * (x[0] - phi[2]) / (pow(phi[1], 3)*r_2_pi);
    hess[2][0] = es * (x[0] - phi[2]) / (pow(phi[1], 3)*r_2_pi);

    hess[0][3] = es * (x[1] - phi[3]) / (pow(phi[1], 3)*r_2_pi);
    hess[3][0] = es * (x[1] - phi[3]) / (pow(phi[1], 3)*r_2_pi);

    hess[1][1] = 2*phi[0]*es*(1 + 5*s + 2*s*s) / ( pow(phi[1], 3) * r_2_pi);

    hess[1][2] = -phi[0] * es * (3 + 2*s) * (x[0] - phi[2]) / (pow(phi[1], 4) * r
      _2_pi);
    hess[1][3] = -phi[0] * es * (3 + 2*s) * (x[1] - phi[3]) / (pow(phi[1], 4) * r
      _2_pi);

    hess[2][1] = hess[1][2];
    hess[3][1] = hess[1][3];

    hess[2][2] = phi[0] * es * (sq(x[0] - phi[2]) - sq(phi[1])) / (r_2_pi * pow(p
      hi[1], 5));
    hess[3][3] = phi[0] * es * (sq(x[1] - phi[3]) - sq(phi[1])) / (r_2_pi * pow(p
      hi[1], 5));

    hess[2][3] = phi[0] * es * (x[0] - phi[2])*(x[1] - phi[3]) / (r_2_pi * pow(ph
      i[1], 5));
    hess[3][2] = hess[2][3];


    return make_tuple(prob, deriv, hess);
}
```

8.5.1.11 template<class B > std::tr1::tuple<double, TooN::Vector<2>, TooN::Matrix<2> > spot_shape_hess (const TooN::Vector< 2 > & *x*, const TooN::Vector< 4, double, B > & *phi*)

Value of the spot, given the parameters and input location.

The spot is described by the following formula:

$$\mu(\mathbf{x},\phi) = \frac{\phi_1}{\phi_2\sqrt{(2\pi)}}\mathbf{e}^{\mathbf{s}},$$

where

$$s = -\frac{(x_1-\phi_3)^2+(x_2-\phi_4)^2}{2\phi_2^2}.$$

This describes a generic blobby spot function of a variable size. The light output can be tuned by varying ϕ_1, and the level of blur can be changed independently by varying ϕ_2. The derivative is:

$$\frac{\partial\mu}{\partial\phi_1} = \frac{1}{\phi_2\sqrt{2\pi}}e^s \quad (34)$$

$$\frac{\partial\mu}{\partial\phi_2} = -\frac{\phi_1}{\phi_2^2\sqrt{2\pi}}e^s(1+2s) \quad (35)$$

And the hessian is:

$$\frac{\partial^2\mu}{\partial\phi_1^2} = 0 \quad (36)$$

$$\frac{\partial^2\mu}{\partial\phi_1\partial\phi_2} = -\frac{1}{\phi_2^2\sqrt{2\pi}}e^s(1+2s) \quad (37)$$

$$\frac{\partial^2\mu}{\partial\phi_2^2} = \frac{2\phi_1}{\phi_2^3\sqrt{2\pi}}e^s(1+5s+2s^2) \quad (38)$$

Parameters

x	$\mathbf{x}$
phi	ϕ

Returns

$\mu(\mathbf{x},\phi)$

Definition at line 124 of file storm.h.

References spot_shape_s().

Referenced by log_probability_spot_hess().

```
{
    double s = spot_shape_s(x, phi);
    double r_2_pi = sqrt(2*M_PI);

    double prob = exp(s) * phi[0]/(phi[1]*r_2_pi);
    TooN::Vector<2> deriv = (exp(s) / (phi[1]*r_2_pi)) * TooN::makeVector(1, -phi
```

```
      [0] * (1 + 2*s)/phi[1]);
    TooN::Matrix<2> hess;

    hess[0][0] = 0;
    hess[0][1] = -exp(s)*(1+2*s) / (phi[1] * phi[1] * r_2_pi);
    hess[1][0] = hess[0][1];
    hess[1][1] = 2*phi[0]*exp(s)*(1 + 5*s + 2*s*s) / ( pow(phi[1], 3) * r_2_pi);

    return std::tr1::make_tuple(prob, deriv, hess);
}
```

8.5.1.12 template<class B > std::pair<double, TooN::Vector<2>> > spot_shape_diff (const TooN::Vector< 2 > & *x*, const TooN::Vector< 4, double, B > & *phi*)

see spot_shape_hess()

Parameters

x	$\mathbf{x}$
phi	ϕ

Returns

$\mu(\mathbf{x}, \phi)$

Definition at line 146 of file storm.h.

References spot_shape_s().

Referenced by log_probability_spot_diff().

```
{
    double s = spot_shape_s(x, phi);
    double r_2_pi = sqrt(2*M_PI);

    double prob = exp(s) * phi[0]/(phi[1]*r_2_pi);
    TooN::Vector<2> deriv = (exp(s) / (phi[1]*r_2_pi)) * TooN::makeVector(1, -phi
      [0] * (1 + 2*s)/phi[1]);
    return std::make_pair(prob, deriv);
}
```

8.5.1.13 template<class B > double spot_shape (const TooN::Vector< 2 > & *x*, const TooN::Vector< 4, double, B > & *phi*)

see spot_shape_hess()

Parameters

x	$\mathbf{x}$
phi	ϕ

Returns

$\mu(\mathbf{x}, \phi)$

Definition at line 162 of file storm.h.

References spot_shape_s().

Referenced by log_probability_spot().

```
{
    double s = spot_shape_s(x, phi);
    double r_2_pi = sqrt(2*M_PI);

    // FIXME FIXME FIXME and don't forget to fix the HESSIAN AND DERIVATIVE
    // Should be:             1/(2 pi s^2)  for two dimensions
    //                            vvvvvvvvvvvvv    http://lol.i.trollyou.com/
    double prob = exp(s) * phi[0]/(phi[1]*r_2_pi);


    return prob;
}
```

8.5.1.14 template<class Base > std::tr1::tuple<double, TooN::Vector<2>, TooN::Matrix<2> > log_probability_spot_hess (const CVD::SubImage< float > & *im*, double *variance*, const TooN::Vector< 4, double, Base > & *spot_parameters*)

Find the log probability of an image patch, assuming zero base-line mean and the given variance.

This function makes use of the spot shape. It is assumed that the centre pixel of the image is at 0,0. Since the noise is Gaussian:

$$P(\text{image}) = \prod_{\mathbf{x}\in\text{pixels}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(I(\mathbf{x})-\mu(\mathbf{x},\phi))^2}{2\sigma^2}} \tag{39}$$

$$\ln P(\text{image}) = \sum_{\mathbf{x}\in\text{pixels}} -\frac{(I(\mathbf{x})-\mu(\mathbf{x},\phi))^2}{2\sigma^2} - \frac{N}{2}\ln 2\pi\sigma^2, \tag{40}$$

where *I* is the image, and *N* is the number of pixels. See also log_probability_no_spot and μ (spot_shape). The derivatives are:

$$\frac{\partial \ln P(I)}{\partial \phi_0} = \frac{1}{\sigma^2}\sum_{\mathbf{x}}(I_{\mathbf{x}}-\mu(\mathbf{x},\phi))\frac{\partial}{\partial \boldsymbol{\phi_0}}\mu(\mathbf{x},\phi) \tag{41}$$

$$\frac{\partial^2 \ln P(I)}{\partial \phi_0 \partial \phi_1} = \frac{1}{\sigma^2}\sum_{\mathbf{x}}(I_{\mathbf{x}}-\mu(\mathbf{x},\phi))\frac{\partial^2}{\partial \boldsymbol{\phi_0}\partial \boldsymbol{\phi_1}}\mu(\mathbf{x},\phi) - \frac{\partial}{\partial \boldsymbol{\phi_0}}\mu(\mathbf{x},\phi)\frac{\partial}{\partial \boldsymbol{\phi_1}}\mu(\mathbf{x},\phi) \tag{42}$$

Parameters

im	Image
variance	σ^2
spot_parameters	ϕ

Returns

The log probability

Definition at line 217 of file storm.h.

References spot_shape_hess(), and sq().

```
{
    using namespace TooN;
    using namespace std::tr1;

    //-1 because if the image is 3x3, ie 0,1,2 then 1,1 is the centre.
    //If it is 2x2, ie 0,1 then .5,.5 is the centre
    Vector<2> centre = makeVector((im.size().x-1) / 2.0, (im.size().y-1) / 2.0);

    double logprob_part=0;
    Vector<2> diff = Zeros;
    Matrix<2> hess = Zeros;
    for(int y=0; y < im.size().y; y++)
        for(int x=0; x < im.size().x; x++)
        {
            Vector<2> d = TooN::makeVector(x, y) - centre;

            double mu;
            Vector<2> diff_mu;
            Matrix<2> hess_mu;
            tie(mu, diff_mu, hess_mu) = spot_shape_hess(d, spot_parameters);

            double e = im[y][x] - mu;

            logprob_part += -sq(e);
            diff         += diff_mu * e;
            hess         += e * hess_mu - diff_mu.as_col() * diff_mu.as_row();
        }
    return make_tuple(   logprob_part / (2*variance) - im.size().area() * log(2*M
      _PI*variance)/2,
                      diff / variance,
                      hess / variance);
}
```

8.5.1.15 template<class Base > std::pair<double, TooN::Vector<2> > log_probability_spot_diff (const CVD::SubImage< float > & *im,* double *variance,* const TooN::Vector< 4, double, Base > & *spot_parameters*)

See log_probability_spot_hess.

Parameters

im	Image
variance	σ^2
spot_-parameters	ϕ

Returns

The log probability

Definition at line 257 of file storm.h.

References spot_shape_diff(), and sq().

```
{
    using namespace TooN;
    using namespace std::tr1;
    using namespace std;
    //-1 because if the image is 3x3, ie 0,1,2 then 1,1 is the centre.
    //If it is 2x2, ie 0,1 then .5,.5 is the centre
    Vector<2> centre = makeVector((im.size().x-1) / 2.0, (im.size().y-1) / 2.0);

    double logprob_part=0;
    Vector<2> diff = Zeros;
    for(int y=0; y < im.size().y; y++)
        for(int x=0; x < im.size().x; x++)
        {
            Vector<2> d = makeVector(x, y) - centre;

            double mu;
            Vector<2> diff_mu;
            tie(mu, diff_mu) = spot_shape_diff(d, spot_parameters);

            double e = im[y][x] - mu;

            logprob_part += -sq(e);
            diff         += diff_mu * e;
        }
    return make_pair(logprob_part / (2*variance) - im.size().area() * log(2*M_PI*
      variance)/2, diff / variance);
}
```

8.5.1.16 template<class Base > double log_probability_spot (const CVD::SubImage< float > & *im,* double *variance,* const TooN::Vector< 4, double, Base > & *spot_parameters*)

See log_probability_spot_hess.

Parameters

im	Image
variance	σ^2
spot_parameters	ϕ

Returns

The log probability

Definition at line 292 of file storm.h.

References spot_shape(), and sq().

```
{
    //-1 because if the image is 3x3, ie 0,1,2 then 1,1 is the centre.
    //If it is 2x2, ie 0,1 then .5,.5 is the centre
    TooN::Vector<2> centre = TooN::makeVector((im.size().x-1) / 2.0, (im.size().y
      -1) / 2.0);

    double logprob_part=0;
    for(int y=0; y < im.size().y; y++)
        for(int x=0; x < im.size().x; x++)
        {
            TooN::Vector<2> d = TooN::makeVector(x, y) - centre;

            double mu = spot_shape(d, spot_parameters);

            double e = im[y][x] - mu;

            logprob_part += -sq(e);
        }
    return logprob_part / (2*variance) - im.size().area() * log(2*M_PI*variance)/
      2;
}
```

8.5.1.17 double log_normal_std (double *mu*, double *sigma*) [inline]

Compute the standard deviation of a log-normal distribution.

See log_normal().

$$\mathrm{Var}[P(x)] = (e^{(}\sigma^2) - 1)e^{(}2 * \mu + \sigma^2) \tag{43}$$

Parameters

sigma	σ
mu	μ

Returns

The standard deviation

Definition at line 324 of file storm.h.

References sq().

Referenced by brightness_motion_limit().

```
{
    return sqrt((exp(sq(sigma)) - 1) * exp(2*mu + sq(sigma)));
}
```

8.5.1.18 double log_normal_mode (double *mu*, double *sigma*) [inline]

Compute the mode of a log-normal distribution.

See log_normal().

$$\mathrm{Mode}[P(x)] = e^{(}\mu - \sigma^2) \tag{44}$$

Parameters

sigma	σ
mu	μ

Returns

The mode

Definition at line 340 of file storm.h.

Referenced by generate_state_parameters_ye_olde(), and FitSpots::try_modifying_model().

```
{
    return exp(mu - sigma * sigma);
}
```

8.5.1.19 double log_log_normal (double *x*, double *mu*, double *sigma*) [inline]

Log-normal distribution.

This is given by:

$$P(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x-\mu)^2}{s\sigma^2}} \tag{45}$$

$$\ln P(x) = -\frac{(\ln x-\mu)^2}{s\sigma^2} - \ln x - \ln\sigma\sqrt{2\pi}. \tag{46}$$

Parameters

x	x
mu	μ
sigma	σ

Definition at line 355 of file storm.h.

References ln(), and sq().

Referenced by NegativeFreeEnergy::compute_with_mask(), and NegativeFreeEnergy::operator()().

```
{
    return -sq(ln(x) - mu) / (2*sq(sigma)) - ln(x) - ln(sigma * sqrt(2*M_PI));
}
```

8.5.1.20 double diff_log_log_normal (double *x*, double *mu*, double *sigma*) [inline]

Derivative of the log of the log-normal distribution:

$$\frac{\partial \ln P(x)}{\partial x} = -\frac{1}{x}\left(1+\frac{\ln x-\mu}{\sigma^2}\right).$$

.

Parameters

x	x
mu	μ
sigma	σ

Definition at line 369 of file storm.h.

References ln(), and sq().

Referenced by FreeEnergyHessian::hessian(), SpotNegProbabilityDiffWithSampledBackground::operator()(), sampled_background_spot_hessian2(), and sampled_background_spot_hessian_ffbs().

```
{
    return -(1 + (ln(x) - mu)/sq(sigma)) / x;
}
```

8.5.1.21 double hess_log_log_normal (double *x*, double *mu*, double *sigma*) [inline]

Second derivative of the log of the log-normal distribution:

$$\frac{\partial^2 \ln P(x)}{\partial x^2} = \frac{1}{x^2}\left(1 + \frac{\ln x - \mu}{\sigma^2} - \frac{1}{\sigma^2}\right).$$

.

Parameters

x	x
mu	μ
sigma	σ

Definition at line 384 of file storm.h.

References ln(), and sq().

Referenced by FreeEnergyHessian::hessian(), sampled_background_spot_hessian2(), sampled_background_spot_hessian_FAKE(), and sampled_background_spot_hessian_ffbs().

```
{
    return (1 + (ln(x) - mu - 1)/sq(sigma)) / sq(x);
}
```

8.6 Storm imagery classes (basic image processing)

Functions

- vector< Image< float > > load_and_preprocess_images2 (const vector< string > &names)
- vector< Image< float > > load_and_preprocess_images (const vector< string > &names)
- vector< Image< float > > load_and_normalize_images (const vector< string > &files)

8.6.1 Function Documentation

8.6.1.1 vector<Image<float> > load_and_preprocess_images2 (const vector< string > & *names*)

Load all images from disk and do the initial preprocessing.

Parameters

names	List of filenames to load.

Returns

preprocessed images.

Definition at line 23 of file storm_imagery.cc.

```
{
    vector<Image<float> > ims;
    //Load images
    for(unsigned int i=0; i < names.size(); i++)
    {
        Image<float> im = img_load(names[i]);
        ims.push_back(im);

        if(ims.back().size() != ims[0].size())
        {
            cerr << "Error with image " << names[i] << ":  all images must be the
   same size!\n";
            exit(1);
        }
    }
    double mean, variance;
    tie(mean, variance) = mean_and_variance(ims);
    {
        for(unsigned int i=0; i < ims.size(); i++)
            transform(ims[i].begin(), ims[i].end(), ims[i].begin(), bind2nd(minus
  <double>(), mean));
        for(unsigned int i=0; i < ims.size(); i++)
            transform(ims[i].begin(), ims[i].end(), ims[i].begin(), bind1st(multi
  plies<double>(), 1/ sqrt(variance)));
    }

    tie(mean, variance) = mean_and_variance(ims);
```

```
cerr << "Rescaled:\n";
cerr << "mean = " << mean << endl;
cerr << "std  = " << sqrt(variance) << endl;



//Normalize...

//Fit the background model
ImageRef size = ims[0].size();
Vector<10> p = Zeros;
p[6]=-3;
p[9]=-4;

Image<Vector<6> > monomials(size);
Image<double> polynomial(size);
for(int yy=0; yy < size.y; yy++)
    for(int xx=0; xx < size.x; xx++)
    {
        double x = xx *2./ size.x -1 ;
        double x2 = x*x;
        double y = yy *2./size.y - 1;
        double y2 = yy;
        monomials[yy][xx] = makeVector(1, x, y, x2, x*y, y2);
    }


for(int i=0;i < 100;i++)
{
    for(int yy=0; yy < size.y; yy++)
        for(int xx=0; xx < size.x; xx++)
            polynomial[yy][xx] = monomials[yy][xx] *  p.slice<0,6>();

    WLS<10> wls;
    for(unsigned int i=0; i < ims.size(); i++)
        for(int yy=0; yy < size.y; yy++)
            for(int xx=0; xx < size.x; xx++)
            {
                double t = i *1. / ims.size();
                double func = polynomial[yy][xx] * (exp(p[6]*t) + p[8]*exp(p[
  9]*t)) + p[7];

                Vector<10> mJ;

                mJ.slice<0,6>() = exp(p[6]*t)* monomials[yy][xx];
                //mJ.slice<3,3>() = Zeros;
                mJ[6] = polynomial[yy][xx] * exp(p[6]*t) * t;
                //mJ[6] = func  * t;
                mJ[7] = 1;

                mJ[8] = polynomial[yy][xx] * exp(p[9]*t);
                mJ[9] = polynomial[yy][xx] * exp(p[9]*t) * t * p[8];

                double err = ims[i][yy][xx] - func;

                double w;


                if(err > 0)
                    w = .01 / (abs(err) + .01);
                else
                    w = 1;
```

```
                wls.add_mJ(func - ims[i][yy][xx], -mJ, w);
            }

        wls.add_prior(10);
        wls.compute();

        p += wls.get_mu();

        cout << p << endl << endl;
    }

    for(unsigned int i=0; i < ims.size(); i++)
        for(int yy=0; yy < size.y; yy++)
            for(int xx=0; xx < size.x; xx++)
            {
                double x = xx *2./ size.x -1 ;
                double x2 = x*x;
                double y = yy *2./size.y - 1;
                double y2 = yy;
                double t = i *1. / ims.size();
                Vector<6> f = makeVector(1, x, y, x2, x*y, y2);

                double func = f * p.slice<0,6>() * (exp(p[6]*t) + p[8]*exp(p[9]*t
      )) + p[7];
                ims[i][yy][xx] -= func;
            }

    tie(mean, variance) = mean_and_variance(ims);

    //A sanity check.
    cerr << "The mean should be small compared to std:\n";
    cerr << "mean = " << mean << endl;
    cerr << "std  = " << sqrt(variance) << endl;

    //Scale by the variance.
    {
        for(unsigned int i=0; i < ims.size(); i++)
            transform(ims[i].begin(), ims[i].end(), ims[i].begin(), bind1st(multi
      plies<double>(), 1/ sqrt(variance)));
    }
    tie(mean, variance) = mean_and_variance(ims);

    cerr << "Rescaled:\n";
    cerr << "mean = " << mean << endl;
    cerr << "std  = " << sqrt(variance) << endl;

    return ims;
}
```

8.6.1.2 vector<Image<float> > load_and_preprocess_images (const vector< string > & *names*)

Load all images from disk and do the initial preprocessing.

Currently this is a high pass filter to make the resultimg images zero mean.

The filter is controlled with the `preprocess.lpf` and `preprocess.skip` Gvars

See also load_and_preprocess_image()

Parameters

names	List of filenames to load.

Returns

preprocessed images.

Definition at line 172 of file storm_imagery.cc.

Referenced by load_and_normalize_images().

```
{
    vector<Image<float> > ims;

    //float wide = GV3::get<float>("preprocess.lpf", 0., -1);
    //bool p = GV3::get<bool>("preprocess.skip", 0, -1);

    for(unsigned int i=0; i < names.size(); i++)
    {
        Image<float> im = img_load(names[i]);

        ims.push_back(preprocess_image(im));

        if(ims.back().size() != ims[0].size())
        {
            cerr << "Error with image " << names[i] << ":  all images must be the
       same size!\n";
            exit(1);
        }
    }
    return ims;
}
```

8.6.1.3 vector<Image<float> > load_and_normalize_images (const vector< string > & *files*)

Wrapper for load_and_preprocess_images() to allow more flexible behaviour.

Parameters

files	List of filenames to load.

Returns

preprocessed images.

Definition at line 252 of file storm_imagery.cc.

References auto_fixed_scaling(), and load_and_preprocess_images().

Referenced by mmain().

```
{
```

```
    //Load the raw data, and then load the spot parameters.
    vector<Image<float> > ims = load_and_preprocess_images(files);
    double mean, variance;
    tie(mean, variance) = mean_and_variance(ims);

    if(GV3::get<bool>("preprocess.fixed_scaling", 0, FATAL_IF_NOT_DEFINED))
    {
        bool skip = GV3::get<bool>("preprocess.skip");
        if(!skip)
        {
            cerr << "WARNING WARNING WARNING WARNING!!!!!!!!!!!!!!!\n";
            cerr << "preprocessing and fixed scaling selected!!!\n";
            exit(1);
        }

        double sub, div;
        if(GV3::get<bool>("preprocess.fixed_scaling.auto", 0, FATAL_IF_NOT_DEFINE
      D))
        {
            double frac = GV3::get<double>("preprocess.fixed_scaling.auto.proport
      ion", 0, FATAL_IF_NOT_DEFINED);
            tie(sub, div) = auto_fixed_scaling(ims, frac);
        }
        else
        {
            sub = GV3::get<double>("preprocess.fixed_scaling.subtract", 0, FATAL_
      IF_NOT_DEFINED);
            div = GV3::get<double>("preprocess.fixed_scaling.divide", 0, FATAL_IF
      _NOT_DEFINED);
        }

        for(unsigned int i=0; i < ims.size(); i++)
            for(Image<float>::iterator j=ims[i].begin(); j != ims[i].end(); j++)
                *j = (*j - sub)/div;
    }
    else
    {
        //A sanity check.
        cerr << "The mean should be small compared to std:\n";
        cerr << "mean = " << mean << endl;
        cerr << "std  = " << sqrt(variance) << endl;

        //Scale by the variance.
        {
            for(unsigned int i=0; i < ims.size(); i++)
                transform(ims[i].begin(), ims[i].end(), ims[i].begin(), bind1st(m
      ultiplies<double>(), 1/ sqrt(variance)));
        }
    }

    tie(mean, variance) = mean_and_variance(ims);

    //A sanity check.
    cerr << "Rescaled:\n";
    cerr << "mean = " << mean << endl;
    cerr << "std  = " << sqrt(variance) << endl;

    return ims;
}
```

8.7 Storm classes specific to multispot processing

Classes

- struct IndexLexicographicPosition< Cmp, First >

  Class for sorting a list of indexes to an array of spots lexicographically according to the 2D positions of the spots.

9 Namespace Documentation

9.1 SampledMultispot Namespace Reference

Classes

- struct SpotWithBackgroundMasked

  This class compute the log-diff-hess probability of a spot, given an image patch and background due to existing spots.
- class GibbsSampler

  Draw samples from the spot states given the spots positions and some data.
- class GibbsSampler2

  Gibbs sampling class which masks spots to reduce computation.

Functions

- double intensity (double i)
- double intensity (const pair< double, Vector< 4 > > &i)
- template<class T >
  void remove_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample)
- template<class T >
  void add_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample)
- template<class T >
  void remove_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- template<class T >
  void add_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- template<class T >
  void remove_spot (vector< vector< double > > &current_sample_intensities, const vector< vector< T > > &spot_intensities, const vector< State > &spot_-sample, const vector< int > &mask)

- template<class T >
  void add_spot (vector< vector< double > > &current_sample_intensities, const vector< vector< T > > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- vector< double > compute_spot_intensity (const vector< ImageRef > &pixels, const Vector< 4 > &params)
- vector< pair< double, Vector< 4 > > > compute_spot_intensity_derivatives (const vector< ImageRef > &pixels, const Vector< 4 > &params)
- vector< tuple< double, Vector< 4 >, Matrix< 4 > > > compute_spot_intensity_hessian (const vector< ImageRef > &pixels, const Vector< 4 > &params)
- vector< int > sequence (int n)

9.1.1 Function Documentation

9.1.1.1 double SampledMultispot::intensity (double *i*) [inline]

Definition at line 46 of file sampled_multispot.h.

Referenced by add_spot(), and remove_spot().

```
{
    return i;
}
```

9.1.1.2 double SampledMultispot::intensity (const pair< double, Vector< 4 > > & *i*) [inline]

Definition at line 51 of file sampled_multispot.h.

```
{
    return i.first;
}
```

9.1.1.3 template<class T > void SampledMultispot::remove_spot (vector< vector< double > > & *current_sample_intensities,* const vector< T > & *spot_intensities,* const vector< State > & *spot_sample*)

Definition at line 59 of file sampled_multispot.h.

References intensity().

Referenced by SampledMultispot::GibbsSampler2::next(), SampledMultispot::GibbsSampler::next(), FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_modifying_-model().

```
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < spot_intensities.size(); p++)
                current_sample_intensities[frame][p] -= intensity(spot_intensitie
      s[p]);
}
```

9.1.1.4 template<class T > void SampledMultispot::add_spot (vector< vector< double > > & *current_sample_intensities,* const vector< T > & *spot_intensities,* const vector< State > & *spot_sample*)

Definition at line 68 of file sampled_multispot.h.

References intensity().

Referenced by SampledMultispot::GibbsSampler2::next(), SampledMultispot::GibbsSampler::next(), and FitSpots::try_modifying_model().

```
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < spot_intensities.size(); p++)
                current_sample_intensities[frame][p] += intensity(spot_intensitie
      s[p]);
}
```

9.1.1.5 template<class T > void SampledMultispot::remove_spot (vector< vector< double > > & *current_sample_intensities,* const vector< T > & *spot_intensities,* const vector< State > & *spot_sample,* const vector< int > & *mask*)

Definition at line 79 of file sampled_multispot.h.

References intensity().

```
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] -= intensity(spot_inte
      nsities[mask[p]]);
}
```

9.1.1.6 template<class T > void SampledMultispot::add_spot (vector< vector< double > > & *current_sample_intensities,* const vector< T > & *spot_intensities,* const vector< State > & *spot_sample,* const vector< int > & *mask*)

Definition at line 88 of file sampled_multispot.h.

References intensity().

```
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] += intensity(spot_inte
      nsities[mask[p]]);
}
```

9.1.1.7 template<class T > void SampledMultispot::remove_spot (vector< vector< double > > & *current_sample_intensities,* const vector< vector< T > > & *spot_intensities,* const vector< State > & *spot_sample,* const vector< int > & *mask*)

Definition at line 99 of file sampled_multispot.h.

References intensity(), and spot_intensities.

```
{
    const int steps = spot_intensities.size();
    const int frames = current_sample_intensities.size();

    for(int frame=0; frame < frames; frame++)
    {
        int s = frame * steps / frames;

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] -= intensity(spot_inte
      nsities[s][mask[p]]);
    }
}
```

9.1.1.8 template<class T > void SampledMultispot::add_spot (vector< vector< double > > & *current_sample_intensities,* const vector< vector< T > > & *spot_intensities,* const vector< State > & *spot_sample,* const vector< int > & *mask*)

Definition at line 115 of file sampled_multispot.h.

References intensity(), and spot_intensities.

```
{
    const int steps = spot_intensities.size();
    const int frames = current_sample_intensities.size();

    for(int frame=0; frame < frames; frame++)
    {
        int s = frame * steps / frames;

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] += intensity(spot_inte
```

```
      nsities[s][mask[p]]);
    }
}
```

9.1.1.9 vector<double> SampledMultispot::compute_spot_intensity (const vector< ImageRef > & *pixels,* const Vector< 4 > & *params*) [inline]

Definition at line 133 of file sampled_multispot.h.

Referenced by NegativeFreeEnergy::compute_with_mask(), FreeEnergyHessian::hessian(), NegativeFreeEnergy::operator()(), FitSpots::optimize_each_spot_in_turn_for_several_passes(), sampled_background_spot_hessian_ffbs(), and FitSpots::try_modifying_model().

```
{
    vector<double> intensities(pixels.size());

    for(unsigned int i=0; i < pixels.size(); i++)
        intensities[i] = spot_shape(vec(pixels[i]), params);

    return intensities;
}
```

9.1.1.10 vector<pair<double, Vector<4> > > SampledMultispot::compute_spot_intensity_derivatives (const vector< ImageRef > & *pixels,* const Vector< 4 > & *params*) [inline]

Definition at line 144 of file sampled_multispot.h.

Referenced by SpotNegProbabilityDiffWithSampledBackground::operator()().

```
{
    vector<pair<double, Vector<4> > > derivatives(pixels.size());

    for(unsigned int i=0; i < pixels.size(); i++)
        derivatives[i] = spot_shape_diff_position(vec(pixels[i]), params);
    return derivatives;
}
```

9.1.1.11 vector<tuple<double, Vector<4>, Matrix<4> > > SampledMultispot::compute_spot_intensity_hessian (const vector< ImageRef > & *pixels,* const Vector< 4 > & *params*) [inline]

Definition at line 153 of file sampled_multispot.h.

Referenced by FreeEnergyHessian::hessian(), sampled_background_spot_hessian2(), sampled_background_spot_hessian_FAKE(), and sampled_background_spot_hessian_ffbs().

```
{
```

```
    vector<tuple<double, Vector<4>, Matrix<4> > > hessian(pixels.size());

    for(unsigned int i=0; i < pixels.size(); i++)
        hessian[i] = spot_shape_hess_position(vec(pixels[i]), params);
    return hessian;
}
```

10 Class Documentation

10.1 AdvancedDialog Class Reference

Control panel which basically presents the .cfg file in a large edit box along with additional necessary things (frame range and pixel size).

Public Member Functions

- boolean dialogItemChanged (GenericDialog gd, java.awt.AWTEvent e)
- double parseDouble (String s)

Package Functions

- AdvancedDialog (String cfg, int nframes_)
- int parseInt (String s)
- TextField getPixelSizeField ()
- TextField getFirstFrameField ()
- TextField getLastFrameField ()
- double getPixelSize ()
- int getFirstFrame ()
- int getLastFrame ()

Package Attributes

- boolean ok = true
- int nframes

10.1.1 Detailed Description

Control panel which basically presents the .cfg file in a large edit box along with additional necessary things (frame range and pixel size).

Also make the user acknowledge that they might be getting into trouble. This is uglier than I would like.

Definition at line 307 of file three_B.java.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 AdvancedDialog::AdvancedDialog (String *cfg,* int *nframes_*) [inline, package]

Definition at line 311 of file three_B.java.

References dialogItemChanged(), and nframes.

```
{
    super("3B Analysis");
    nframes=nframes_;

    addMessage("Advanced configuration");
    addMessage("                                ");
    addCheckbox("I understand 3B enough to be editing the text below", false)
  ;
    addTextAreas(cfg, null, 40, 100);

    addNumericField("Pixel size (nm / pixel)", 100., 1, 10, "nm");
    addNumericField("First frame", 0., 0, 10, "");
    addNumericField("Last frame", nframes-1., 0, 10, "");
    addDialogListener(this);

    dialogItemChanged(null, null);

    showDialog();
}
```

10.1.3 Member Function Documentation

10.1.3.1 boolean AdvancedDialog::dialogItemChanged (GenericDialog *gd,* java.awt.AWTEvent *e*) [inline]

Definition at line 331 of file three_B.java.

References getFirstFrame(), getFirstFrameField(), getLastFrame(), getLastFrameField(), getPixelSizeField(), nframes, and ok.

Referenced by AdvancedDialog().

```
{
    boolean v = ((Checkbox)(getCheckboxes().get(0))).getState();

    if(v != ok)
    {
        ok=v;

        if(ok)
        {
            getTextArea1().setEditable(true);
            ((Label)getMessage()).setText("Warning: strange behaviour may res
  ult!");
```

```
            getPixelSizeField().setEditable(true);
            getFirstFrameField().setEditable(true);
            getLastFrameField().setEditable(true);

        }
        else
        {
            getTextArea1().setEditable(false);
            ((Label)getMessage()).setText("
             ");
            getPixelSizeField().setEditable(false);
            getFirstFrameField().setEditable(false);
            getLastFrameField().setEditable(false);
        }
    }


    //Clamp the frames
    int first = getFirstFrame();
    int last  = getLastFrame();

    int nfirst = Math.max(0, Math.min(nframes-1, first));
    int nlast = Math.max(nfirst, Math.min(nframes-1, last));

    if(first != nfirst)
        getFirstFrameField().setText(Integer.toString(nfirst));
    if(last != nlast)
        getLastFrameField().setText(Integer.toString(nlast));
    return ok;
}
```

10.1.3.2 int AdvancedDialog::parseInt (String *s*) [inline, package]

Definition at line 373 of file three_B.java.

Referenced by getFirstFrame(), and getLastFrame().

```
{
    try
    {
        return Integer.parseInt(s);
    }
    catch(Exception e)
    {
        return 0;
    }
}
```

10.1.3.3 double AdvancedDialog::parseDouble (String *s*) [inline]

Definition at line 385 of file three_B.java.

Referenced by getPixelSize().

```
{
```

```
        try
        {
            return Double.parseDouble(s);
        }
        catch(Exception e)
        {
            return 0;
        }
    }
```

10.1.3.4 TextField AdvancedDialog::getPixelSizeField ( ) [inline, package]

Definition at line 397 of file three_B.java.

Referenced by dialogItemChanged(), and getPixelSize().

```
    {
        return (TextField)(getNumericFields().get(0));
    }
```

10.1.3.5 TextField AdvancedDialog::getFirstFrameField ( ) [inline, package]

Definition at line 402 of file three_B.java.

Referenced by dialogItemChanged(), and getFirstFrame().

```
    {
        return (TextField)(getNumericFields().get(1));
    }
```

10.1.3.6 TextField AdvancedDialog::getLastFrameField ( ) [inline, package]

Definition at line 408 of file three_B.java.

Referenced by dialogItemChanged(), and getLastFrame().

```
    {
        return (TextField)(getNumericFields().get(2));
    }
```

10.1.3.7 double AdvancedDialog::getPixelSize ( ) [inline, package]

Definition at line 413 of file three_B.java.

References getPixelSizeField(), and parseDouble().

```
    {
        return parseDouble(getPixelSizeField().getText());
    }
```

10.1.3.8 int AdvancedDialog::getFirstFrame ( ) [inline, package]

Definition at line 417 of file three_B.java.

References getFirstFrameField(), and parseInt().

Referenced by dialogItemChanged().

```
{
    return parseInt(getFirstFrameField().getText());
}
```

10.1.3.9 int AdvancedDialog::getLastFrame ( ) [inline, package]

Definition at line 422 of file three_B.java.

References getLastFrameField(), and parseInt().

Referenced by dialogItemChanged().

```
{
    return parseInt(getLastFrameField().getText());
}
```

10.1.4 Member Data Documentation

10.1.4.1 boolean AdvancedDialog::ok = true [package]

Definition at line 309 of file three_B.java.

Referenced by dialogItemChanged().

10.1.4.2 int AdvancedDialog::nframes [package]

Definition at line 310 of file three_B.java.

Referenced by AdvancedDialog(), and dialogItemChanged().

The documentation for this class was generated from the following file:

- three_B.java

10.2 ClassicGlow Class Reference

Plugin implementing the classic glow/hot LUT which seems to be missing from ImageJ.

Static Package Attributes

- static byte red []
- static byte green []
- static byte blue []

10.2.1 Detailed Description

Plugin implementing the classic glow/hot LUT which seems to be missing from ImageJ.

Definition at line 12 of file ClassicGlow.java.

10.2.2 Member Data Documentation

10.2.2.1 byte ClassicGlow::red[] [static, package]

Definition at line 14 of file ClassicGlow.java.

10.2.2.2 byte ClassicGlow::green[] [static, package]

Definition at line 15 of file ClassicGlow.java.

10.2.2.3 byte ClassicGlow::blue[] [static, package]

Definition at line 16 of file ClassicGlow.java.

The documentation for this class was generated from the following file:

- ClassicGlow.java

10.3 CloseButtonListener Class Reference

Close button issues a window close event.

Public Member Functions

- CloseButtonListener (JFrame fr)
- void actionPerformed (ActionEvent e)

Private Attributes

- JFrame f

10.3.1 Detailed Description

Close button issues a window close event.

Actual closing logic is then done in the close event handler.

Definition at line 837 of file three_B.java.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 CloseButtonListener::CloseButtonListener (JFrame *fr*) [inline]

Definition at line 840 of file three_B.java.

References f.

```
{
    f = fr;
}
```

10.3.3 Member Function Documentation

10.3.3.1 void CloseButtonListener::actionPerformed (ActionEvent *e*) [inline]

Definition at line 846 of file three_B.java.

References f.

```
{
    //Voodoo from Stack Overflow
    WindowEvent wev = new WindowEvent(f, WindowEvent.WINDOW_CLOSING);
    Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(wev);
}
```

10.3.4 Member Data Documentation

10.3.4.1 JFrame CloseButtonListener::f [private]

Definition at line 839 of file three_B.java.

Referenced by actionPerformed(), and CloseButtonListener().

The documentation for this class was generated from the following file:

- three_B.java

10.4 ConjugateGradientOnly< Size, Precision > Struct Template Reference

Class for performing optimization with Conjugate Gradient, where only the derivatives are available.

`#include <conjugate_gradient_only.h>`

Public Member Functions

- template<class Deriv >
  ConjugateGradientOnly (const TooN::Vector< Size > &start, const Deriv &deriv, const TooN::Vector< Size > &d)
- template<int S, class P , class B >
  void init (const TooN::Vector< Size > &start, const TooN::Vector< S, P, B > &deriv)
- template<class Deriv >
  void init (const TooN::Vector< Size > &start, const Deriv &deriv)
- template<class Deriv >
  void find_next_point (const Deriv &deriv)
- bool finished ()
- void update_vectors_PR (const TooN::Vector< Size > &grad)
- template<class Deriv >
  bool iterate (const Deriv &deriv)

Public Attributes

- const int size
- TooN::Vector< Size > g
- TooN::Vector< Size > new_g
- TooN::Vector< Size > h
- TooN::Vector< Size > minus_h
- TooN::Vector< Size > old_g
- TooN::Vector< Size > old_h
- TooN::Vector< Size > x
- TooN::Vector< Size > old_x
- Precision tolerance
- int max_iterations
- TooN::Vector< Size > delta_max
- Precision linesearch_tolerance
- Precision linesearch_epsilon
- int iterations

10.4.1 Detailed Description

template<int Size = -1, class Precision = double>struct ConjugateGradientOnly< Size, Precision >

Class for performing optimization with Conjugate Gradient, where only the derivatives are available.

Definition at line 9 of file conjugate_gradient_only.h.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 template<int Size = -1, class Precision = double> template<class Deriv > ConjugateGradientOnly< Size, Precision >::ConjugateGradientOnly (const TooN::Vector< Size > & *start,* const Deriv & *deriv,* const TooN::Vector< Size > & *d*) [inline]

Initialize the ConjugateGradient class with sensible values.

Parameters

start	Starting point, x
deriv	Function to compute $\nabla f(x)$
d	Maximum allowed movement (delta) in any dimension

Definition at line 36 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::init().

```
: size(start.size()),
  g(size),new_g(size),h(size),minus_h(size),old_g(size),old_h(size),x(start),
  old_x(size),delta_max(d)
{
    init(start, deriv);
}
```

10.4.3 Member Function Documentation

10.4.3.1 template<int Size = -1, class Precision = double> template<int S, class P , class B > void ConjugateGradientOnly< Size, Precision >::init (const TooN::Vector< Size > & *start,* const TooN::Vector< S, P, B > & *deriv*) [inline]

Initialise given a starting point and the derivative at the starting point.

Parameters

start	starting point
deriv	derivative at starting point

Definition at line 46 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::g, ConjugateGradientOnly< Size, Precision >::h, ConjugateGradientOnly< Size, Precision >::iterations, ConjugateGradientOnly< Size, Precision >::linesearch_epsilon, ConjugateGradientOnly< Size, Precision >::linesearch_tolerance, ConjugateGradientOnly< Size, Precision >::max_iterations, ConjugateGradientOnly< Size, Precision >::minus_h, ConjugateGradientOnly< Size, Precision >::size, ConjugateGradientOnly< Size, Precision >::tolerance, and ConjugateGradientOnly< Size, Precision >::x.

Referenced by ConjugateGradientOnly< Size, Precision >::ConjugateGradientOnly(), ConjugateGradientOnly< Size, Precision >::init(), and FitSpots::try_modifying_model().

```
{

    using std::numeric_limits;
    x = start;

    //Start with the conjugate direction aligned with
    //the gradient
    g = deriv;
    h = g;
    minus_h=-h;

    tolerance = sqrt(numeric_limits<Precision>::epsilon());
    max_iterations = size * 100;


    linesearch_tolerance =  sqrt(numeric_limits<Precision>::epsilon());
    linesearch_epsilon = 1e-20;

    iterations=0;
}
```

10.4.3.2 template<int Size = -1, class Precision = double> template<class Deriv > void ConjugateGradientOnly< Size, Precision >::init (const TooN::Vector< Size > & *start,* const Deriv & *deriv*) [inline]

Initialise given a starting point and a functor for computing derivatives.

Parameters

start	starting point
deriv	derivative computing functor

Definition at line 71 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::init().

```
{
    init(start, deriv(start));
}
```

10.4.3.3 template<int Size = -1, class Precision = double> template<class Deriv > void ConjugateGradientOnly< Size, Precision >::find_next_point (const Deriv & *deriv*) [inline]

Perform a linesearch from the current point (x) along the current conjugate vector (h).

The linesearch does not make use of values. You probably do not want to call this function directly. See iterate() instead. This function updates:

- x
- old_c
- iterations Note that the conjugate direction and gradient are not updated. If bracket_minimum_forward detects a local maximum, then essentially a zero sized step is taken.

Parameters

deriv	Functor returning the derivative value at a given point.

Definition at line 87 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::delta_max, ConjugateGradientOnly< Size, Precision >::g, ConjugateGradientOnly< Size, Precision >::h, ConjugateGradientOnly< Size, Precision >::iterations, ConjugateGradientOnly< Size, Precision >::linesearch_epsilon, ConjugateGradientOnly< Size, Precision >::linesearch_tolerance, ConjugateGradientOnly< Size, Precision >::minus_h, ConjugateGradientOnly< Size, Precision >::new_g, and ConjugateGradientOnly< Size, Precision >::x.

Referenced by ConjugateGradientOnly< Size, Precision >::iterate().

```
{
    iterations++;
    using std::abs;
    //Conjugate direction is -h
    //grad.-h is (should be negative)

    //We should have that f1 is negative.
    new_g = g;
    double f1 = g * minus_h;
    //If f1 is positive, then the conjugate vector points agains the
    //newly computed gradient. This can only happen if there is an error
    //in the computation of the gradient (eg we're using a stochastic method)

    //not much to do here except to stop.
    if(f1 > 0)
    {
        //Return, so try to search in a direction conjugate to the current on
  e.
        return;
    }
    //What step size takes us up to the maximum length
    Precision max_step = HUGE_VAL;
    for(int i=0; i < minus_h.size(); i++)
        max_step = min(max_step, abs(delta_max[i]/h[i]));

    //Taking the maximum step size may result in NaNs.
```

```
        //Try the maximum step size, and seccessively reduce it.
        Precision full_max_step = max_step;

        for(;;)
        {
            new_g = deriv(x + max_step * minus_h);

            if(!TooN::isnan(new_g))
            {
//  cout << "new_g is NAN free :)\n";
                break;
            }
            else
                max_step /=2;

            //If the step size gets too small then
            //return as we can't really do anything
            if(max_step < full_max_step * linesearch_tolerance)
                return;
        }

        double f2 = new_g * minus_h;


        //If f2 hasn't gone negative, then the largest allowed step is not large
      enough.
        //So, take a full step, then keep going in the same direction
        if(f2 < 0)
        {
            //Take a full step
            x += max_step * minus_h;
            return;
        }

        //Now, x1 and x2 bracket a root, and find the root using bisection
        //x1 and x2 are represented by the scalars s1 and s2
        double s1 = 0;
        double s2 = max_step;
        double s_new = max_step;

        int updates[2] = {0,0};

        while(abs(s1 - s2) > abs(s1 + s2) * linesearch_tolerance +
      linesearch_epsilon)
        {
            if(updates[0] != updates[1] && updates[0] != 0)
            {

                //Compute the new point using false position.
                s_new = s1 + f1 * (s2 - s1)/(f1 - f2);
                new_g = deriv(x + s_new * minus_h);
                double f_new = new_g*minus_h;

                updates[0] = updates[1];

                if(f_new == 0)
                    break;

                if(f_new < 0)
                {
                    s1 = s_new;
                    f1 = f_new;
```

```
                updates[1] = 1;
            }
            else
            {
                s2 = s_new;
                f2 = f_new;
                updates[1] = 2;
            }
        }
        else
        {
            //Compute the new point

            s_new = (s1 + s2) / 2;

            new_g = deriv(x + s_new * minus_h);
            double f_new = new_g*minus_h;

            if(f_new < 0)
            {
                s1 = s_new;
                f1 = f_new;
                updates[0] = 1;
            }
            else
            {
                s2 = s_new;
                f2 = f_new;
                updates[0] = 2;
            }

        }

    }

    //Update the current position and value
    //The most recent gradient computation is at s_new
    x += minus_h * s_new;

}
```

10.4.3.4 template<int Size = -1, class Precision = double> bool ConjugateGradientOnly< Size, Precision >::finished ( ) [inline]

Check to see it iteration should stop.

You probably do not want to use this function. See iterate() instead. This function updates nothing.

Definition at line 215 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::iterations, ConjugateGradientOnly< Size, Precision >::max_iterations, ConjugateGradientOnly< Size, Precision >::new_g, and ConjugateGradientOnly< Size, Precision >::tolerance.

Referenced by ConjugateGradientOnly< Size, Precision >::iterate().

```
{
```

```
    using std::abs;
    return iterations > max_iterations || norm_inf(new_g) < tolerance;
}
```

10.4.3.5 template<int Size = -1, class Precision = double> void ConjugateGradientOnly< Size, Precision >::update_vectors_PR (const TooN::Vector< Size > & *grad*) [inline]

After an iteration, update the gradient and conjugate using the Polak-Ribiere equations.

This function updates:

- g
- old_g
- h
- old_h

Parameters

grad	The derivatives of the function at *x*

Definition at line 229 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::g, ConjugateGradientOnly< Size, Precision >::h, ConjugateGradientOnly< Size, Precision >::minus_h, ConjugateGradientOnly< Size, Precision >::old_g, and ConjugateGradientOnly< Size, Precision >::old_h.

Referenced by ConjugateGradientOnly< Size, Precision >::iterate().

```
{
    //Update the position, gradient and conjugate directions
    old_g = g;
    old_h = h;

    g = grad;
    //Precision gamma = (g * g - oldg*g)/(oldg * oldg);
    Precision gamma = (g * g - old_g*g)/(old_g * old_g);
    h = g + gamma * old_h;
    minus_h=-h;
}
```

10.4.3.6 template<int Size = -1, class Precision = double> template<class Deriv > bool ConjugateGradientOnly< Size, Precision >::iterate (const Deriv & *deriv*) [inline]

Use this function to iterate over the optimization.

Note that after iterate returns false, g, h, old_g and old_h will not have been updated. This function updates:

- x
- old_c
- iterations
- g*
- old_g*
- h*
- old_h* 'd variables not updated on the last iteration.

Parameters

deriv	Functor to compute derivatives at the specified point.

Returns

Whether to continue.

Definition at line 256 of file conjugate_gradient_only.h.

References ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::finished(), ConjugateGradientOnly< Size, Precision >::new_g, and ConjugateGradientOnly< Size, Precision >::update_vectors_PR().

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_-modifying_model().

```
{
    find_next_point(deriv);

    if(!finished())
    {
        update_vectors_PR(new_g);
        return 1;
    }
    else
        return 0;
}
```

10.4.4 Member Data Documentation

10.4.4.1 template<int Size = -1, class Precision = double> const int ConjugateGradientOnly< Size, Precision >::size

Dimensionality of the space.

Definition at line 11 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::init().

10.4.4.2 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::g

Gradient vector used by the next call to iterate()

Definition at line 12 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::init(), and ConjugateGradientOnly< Size, Precision >::update_vectors_-PR().

10.4.4.3 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::new_g

The new gradient set at the end of iterate()

Definition at line 13 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::finished(), and ConjugateGradientOnly< Size, Precision >::iterate().

10.4.4.4 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::h

Conjugate vector to be searched along in the next call to iterate()

Definition at line 14 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::init(), and ConjugateGradientOnly< Size, Precision >::update_vectors_-PR().

10.4.4.5 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::minus_h

negative of h as this is required to be passed into a function which uses references (so can't be temporary)

Definition at line 15 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::init(), and ConjugateGradientOnly< Size, Precision >::update_vectors_-PR().

10.4.4.6 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::old_g

Gradient vector used to compute h in the last call to iterate()

Definition at line 16 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::update_vectors_PR().

10.4.4.7 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::old_h

Conjugate vector searched along in the last call to iterate()

Definition at line 17 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::update_vectors_PR().

10.4.4.8 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::x

Current position (best known point)

Definition at line 18 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::init(), FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_modifying_model().

10.4.4.9 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::old_x

Previous point (not set at construction)

Definition at line 19 of file conjugate_gradient_only.h.

10.4.4.10 template<int Size = -1, class Precision = double> Precision ConjugateGradientOnly< Size, Precision >::tolerance

Tolerance used to determine if the optimization is complete. Defaults to square root of machine precision.

Definition at line 21 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::finished(), and ConjugateGradientOnly< Size, Precision >::init().

10.4.4.11 template<int Size = -1, class Precision = double> int ConjugateGradientOnly< Size, Precision >::max_iterations

Maximum number of iterations. Defaults to `size` $*100$.

Definition at line 22 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::finished(), ConjugateGradientOnly<

Size, Precision >::init(), and FitSpots::optimize_each_spot_in_turn_for_several_passes().

10.4.4.12 template<int Size = -1, class Precision = double> TooN::Vector<Size> ConjugateGradientOnly< Size, Precision >::delta_max

Maximum distance to travel along all axes in line search.

Definition at line 24 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point().

10.4.4.13 template<int Size = -1, class Precision = double> Precision ConjugateGradientOnly< Size, Precision >::linesearch_tolerance

Tolerance used to determine if the linesearch is complete. Defaults to square root of machine precision.

Definition at line 26 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), and ConjugateGradientOnly< Size, Precision >::init().

10.4.4.14 template<int Size = -1, class Precision = double> Precision ConjugateGradientOnly< Size, Precision >::linesearch_epsilon

Additive term in tolerance to prevent excessive iterations if $x_{\text{optimal}} = 0$. Known as ZEPS in numerical recipies. Defaults to 1e-20.

Definition at line 27 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), and ConjugateGradientOnly< Size, Precision >::init().

10.4.4.15 template<int Size = -1, class Precision = double> int ConjugateGradientOnly< Size, Precision >::iterations

Number of iterations performed.

Definition at line 30 of file conjugate_gradient_only.h.

Referenced by ConjugateGradientOnly< Size, Precision >::find_next_point(), ConjugateGradientOnly< Size, Precision >::finished(), and ConjugateGradientOnly< Size, Precision >::init().

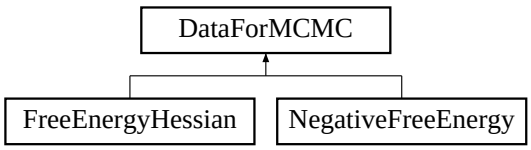
The documentation for this struct was generated from the following file:

- conjugate_gradient_only.h

10.5 DataForMCMC Class Reference

Closure hoding the data required do use GibbsSampler2 See FitSpots for naming of variables.

Inheritance diagram for DataForMCMC:



Public Member Functions

- MT19937 & get_rng () const
- DataForMCMC (const vector< ImageRef > &pixels_, const vector< vector< double > > &pixel_intensities_, double mu_brightness_, double sigma_brightness_, double mu_blur_, double sigma_blur_, double variance_, int samples_, int sample_iterations_, Matrix< 3 > A_, Vector< 3 > pi_, MT19937 &rng_)

Protected Attributes

- const vector< ImageRef > & pixels
- const vector< vector< double > > & pixel_intensities
- const double mu_brightness
- const double sigma_brightness
- const double mu_blur
- const double sigma_blur
- const double variance
- const int samples
- const int sample_iterations
- const Matrix< 3 > A
- const Vector< 3 > pi
- MT19937 & rng

10.5.1 Detailed Description

Closure hoding the data required do use GibbsSampler2 See FitSpots for naming of variables.

Definition at line 176 of file multispot5.cc.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 DataForMCMC::DataForMCMC (const vector< ImageRef > & *pixels_*, const vector< vector< double > > & *pixel_intensities_*, double *mu_brightness_*, double *sigma_brightness_*, double *mu_blur_*, double *sigma_blur_*, double *variance_*, int *samples_*, int *sample_iterations_*, Matrix< 3 > *A_*, Vector< 3 > *pi_*, MT19937 & *rng_*) [inline]

Definition at line 195 of file multispot5.cc.

```
:pixels(pixels_),
 pixel_intensities(pixel_intensities_),
 mu_brightness(mu_brightness_),
 sigma_brightness(sigma_brightness_),
 mu_blur(mu_blur_),
 sigma_blur(sigma_blur_),
 variance(variance_),
 samples(samples_),
 sample_iterations(sample_iterations_),
 A(A_),
 pi(pi_),
 rng(rng_)
{}
```

10.5.3 Member Function Documentation

10.5.3.1 MT19937& DataForMCMC::get_rng ( ) const [inline]

Definition at line 189 of file multispot5.cc.

Referenced by NegativeFreeEnergy::compute_with_mask(), and NegativeFreeEnergy::operator()().

```
{
    return rng;
}
```

10.5.4 Member Data Documentation

10.5.4.1 const vector<ImageRef>& DataForMCMC::pixels [protected]

Definition at line 179 of file multispot5.cc.

10.5.4.2 const vector<vector<double> >& DataForMCMC::pixel_intensities [protected]

Definition at line 180 of file multispot5.cc.

10.5.4.3 const double DataForMCMC::mu_brightness [protected]

Definition at line 181 of file multispot5.cc.

10.5.4.4 const double DataForMCMC::sigma_brightness [protected]

Definition at line 181 of file multispot5.cc.

10.5.4.5 const double DataForMCMC::mu_blur [protected]

Definition at line 181 of file multispot5.cc.

10.5.4.6 const double DataForMCMC::sigma_blur [protected]

Definition at line 181 of file multispot5.cc.

10.5.4.7 const double DataForMCMC::variance [protected]

Definition at line 182 of file multispot5.cc.

10.5.4.8 const int DataForMCMC::samples [protected]

Definition at line 183 of file multispot5.cc.

10.5.4.9 const int DataForMCMC::sample_iterations [protected]

Definition at line 183 of file multispot5.cc.

10.5.4.10 const Matrix<3> DataForMCMC::A [protected]

Definition at line 184 of file multispot5.cc.

10.5.4.11 const Vector<3> DataForMCMC::pi [protected]

Definition at line 185 of file multispot5.cc.

10.5.4.12 MT19937& DataForMCMC::rng [protected]

Definition at line 186 of file multispot5.cc.

The documentation for this class was generated from the following file:

- multispot5.cc

10.6 EControlPanel Class Reference

Control panel for running 3B plugin and providing interactive update.

Public Member Functions

- void windowOpened (WindowEvent e)
- void windowClosed (WindowEvent e)
- void windowDeactivated (WindowEvent e)
- void windowActivated (WindowEvent e)
- void windowDeiconified (WindowEvent e)
- void windowIconified (WindowEvent e)
- void windowClosing (WindowEvent e)
- void issue_stop ()
- synchronized void append (final ArrayList< Spot > p, int its)
- void send_update_canvas_event ()
- void die (final String s)
- void send_status_text_message (final String s)
- void send_time_text_message (final String s)

Package Functions

- EControlPanel (Rectangle roi_, double ps_, String filename_, ThreeBRunner tbr_-
)
- void export_reconstruction_as_ij ()

Package Attributes

- JButton exportButton
- JButton closeButton
- JLabel time_msg

Private Member Functions

- GridBagConstraints canvas_pos ()
- GridBagConstraints status_pos ()
- GridBagConstraints time_pos ()
- GridBagConstraints buttons_pos ()

- GridBagConstraints pixel_size_pos ()
- GridBagConstraints blur_pos ()
- void set_status (String s)
- void set_time (String s)
- synchronized void update_canvas ()
- synchronized FloatProcessor reconstruct ()

Private Attributes

- ImagePlus linear_reconstruction
- ImageCanvas canvas
- JButton stopButton
- Color exportButtonColor
- JLabel status
- FloatSliderWithBox blur_fwhm
- FloatSliderWithBox reconstructed_pixel_size
- Panel complete
- JPanel buttons
- ThreeBRunner tbr
- ArrayList< Spot > pts
- Rectangle roi
- double zoom = 0.01
- double reconstruction_blur_fwhm = .5
- double pixel_size_in_nm
- String filename
- int iterations = 0

10.6.1 Detailed Description

Control panel for running 3B plugin and providing interactive update.

Definition at line 888 of file three_B.java.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 EControlPanel::EControlPanel (Rectangle *roi_*, double *ps_*, String *filename_*, ThreeBRunner *tbr_*) [inline, package]

Definition at line 918 of file three_B.java.

References FloatSliderWithBox::addChangeListener(), blur_fwhm, blur_pos(), buttons, buttons_pos(), canvas, canvas_pos(), closeButton, complete, exportButton, exportButtonColor, filename, linear_reconstruction, pixel_size_in_nm, pixel_size_pos(), pts, reconstruct(), reconstructed_pixel_size, ThreeBRunner::register(), roi, set_status(), set_time(), FloatSliderWithBox::setFormat(), FloatSliderWithBox::setUnits(), status, status_pos(), stopButton, tbr, time_msg, and time_pos().

```
{
    //Constract superclass
    super(filename_);

    tbr = tbr_;
    filename=filename_;

    roi = roi_;
    pixel_size_in_nm = ps_;
    pts = new ArrayList<Spot>();


    //Now generate the dialog box
    complete = new Panel(new GridBagLayout());

    //Create the image viewer
    linear_reconstruction = new ImagePlus();
    linear_reconstruction.setProcessor(reconstruct());
    canvas = new ImageCanvas(linear_reconstruction);

    //Make the image scrollable. This seems to work, if the correct cargo-cul
  ting
    //is performed with the gridbaglayout fill constraints. I don't really un
  derstand why.
    ScrollPane scroll = new ScrollPane();
    scroll.add(canvas);
    complete.add(scroll, canvas_pos());

    //Create the status message
    status = new JLabel();
    if(tbr != null)
        set_status("Running.");
    else
        set_status("Using loaded data.");
    complete.add(status, status_pos());

    //Create the ETA massage
    time_msg = new JLabel();
    if(tbr != null)
        set_time("unknown");
    else
        set_time("not running");
    complete.add(time_msg, time_pos());



    //The two control sliders
    blur_fwhm = new FloatSliderWithBox("Reconstruction blur FWHM", 0, 150, 10
  0.0, 5, false);
    blur_fwhm.setUnits("nm").setFormat("%5.1f");
    blur_fwhm.addChangeListener(new SomethingChanges(this));
    complete.add(blur_fwhm, blur_pos());

    reconstructed_pixel_size = new FloatSliderWithBox("Reconstructed pixel si
  ze", 1, 300, 10.0, 5, true);
    reconstructed_pixel_size.setUnits("nm").setFormat("%5.1f");
    reconstructed_pixel_size.addChangeListener(new SomethingChanges(this));
    complete.add(reconstructed_pixel_size, pixel_size_pos());

    //The button bar
    buttons = new JPanel();
```

```
        exportButton = new JButton("Export...");
        exportButton.addActionListener(new ExportButtonListener(this));
        buttons.add(exportButton);
        exportButtonColor = exportButton.getBackground();

        //No point in having a stop button if this is just a viewer
        if(tbr != null)
        {
            stopButton = new JButton("Stop");
            stopButton.addActionListener(new StopButtonListener(this));
            buttons.add(stopButton);
        }

        closeButton = new JButton("Close");
        closeButton.addActionListener(new CloseButtonListener(this));
        buttons.add(closeButton);


        complete.add(buttons, buttons_pos());

        getContentPane().add(complete);

        //This class knows how to handle window close events.
        //Don't close the window by default, so we can try to bring up a
        //confirmation dialog.
        addWindowListener(this);
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);


        //Cargo culting...
        pack();
        setVisible(true);
        validate();


        //Start the 3B thread, if we have one to run.
        if(tbr != null)
        {
            tbr.register(this);
            Thread t = new Thread(tbr);
            t.start();
        }
    }
```

10.6.3 Member Function Documentation

10.6.3.1 GridBagConstraints EControlPanel::canvas_pos ( ) [inline, private]

Definition at line 1022 of file three_B.java.

Referenced by EControlPanel().

```
    {
        GridBagConstraints g = new GridBagConstraints();
        g.gridx = 0;
        g.gridy = 0;
        g.fill = GridBagConstraints.BOTH;
```

```
        g.weightx=100;
        g.weighty=100;
        return g;
    }
```

10.6.3.2 GridBagConstraints EControlPanel::status_pos ( ) [inline, private]

Definition at line 1033 of file three_B.java.

Referenced by EControlPanel().

```
    {
        GridBagConstraints g = new GridBagConstraints();
        g.gridx = 0;
        g.gridy = 1;
        g.anchor = GridBagConstraints.FIRST_LINE_START;
        g.fill = GridBagConstraints.BOTH;
        return g;
    }
```

10.6.3.3 GridBagConstraints EControlPanel::time_pos ( ) [inline, private]

Definition at line 1043 of file three_B.java.

Referenced by EControlPanel().

```
    {
        GridBagConstraints g = new GridBagConstraints();
        g.gridx = 0;
        g.gridy = 2;
        g.anchor = GridBagConstraints.FIRST_LINE_START;
        g.fill = GridBagConstraints.BOTH;
        return g;
    }
```

10.6.3.4 GridBagConstraints EControlPanel::buttons_pos ( ) [inline, private]

Definition at line 1052 of file three_B.java.

Referenced by EControlPanel().

```
    {
        GridBagConstraints g = new GridBagConstraints();
        g.gridx = 0;
        g.gridy = 5;
        g.fill = GridBagConstraints.BOTH;
        return g;
    }
```

10.6.3.5 GridBagConstraints EControlPanel::pixel_size_pos ( ) [inline, private]

Definition at line 1061 of file three_B.java.

Referenced by EControlPanel().

```
{
    GridBagConstraints g = new GridBagConstraints();
    g.gridx = 0;
    g.gridy = 4;
    g.anchor = GridBagConstraints.FIRST_LINE_START;
    g.fill = GridBagConstraints.BOTH;
    return g;
}
```

10.6.3.6 GridBagConstraints EControlPanel::blur_pos ( ) [inline, private]

Definition at line 1071 of file three_B.java.

Referenced by EControlPanel().

```
{
    GridBagConstraints g = new GridBagConstraints();
    g.gridx = 0;
    g.gridy = 3;
    g.anchor = GridBagConstraints.FIRST_LINE_START;
    g.fill = GridBagConstraints.BOTH;
    return g;
}
```

10.6.3.7 void EControlPanel::set_status (String *s*) [inline, private]

Definition at line 1081 of file three_B.java.

References status.

Referenced by EControlPanel(), and send_status_text_message().

```
{
    status.setText("Status: " + s);
}
```

10.6.3.8 void EControlPanel::set_time (String *s*) [inline, private]

Definition at line 1086 of file three_B.java.

References time_msg.

Referenced by EControlPanel(), and send_time_text_message().

```
{
    time_msg.setText("Estimated time: " + s);
}
```

10.6.3.9 synchronized void EControlPanel::update_canvas ( ) [inline, private]

Generic redraw function which recomputes the reconstruction.

this is a synchronized method since it makes use of the shared writable array of points.

Definition at line 1094 of file three_B.java.

References blur_fwhm, canvas, ThreeBGlobalConstants::critical_iterations, exportButton, exportButtonColor, FloatSliderWithBox::getValue(), iterations, linear_reconstruction, pixel_size_in_nm, reconstruct(), reconstructed_pixel_size, reconstruction_blur_fwhm, and zoom.

Referenced by send_update_canvas_event().

```
{
    reconstruction_blur_fwhm = blur_fwhm.getValue();
    zoom = pixel_size_in_nm / reconstructed_pixel_size.getValue();

    linear_reconstruction.setProcessor(reconstruct());
    linear_reconstruction.updateImage();
    linear_reconstruction.updateAndRepaintWindow();
    canvas.repaint();
    //Update the canvas size and viewport to prevent funny things with zoomin
  g.
    canvas.setDrawingSize(linear_reconstruction.getWidth(),
  linear_reconstruction.getHeight());
    canvas.setSourceRect(new Rectangle(linear_reconstruction.getWidth(),
  linear_reconstruction.getHeight()));
    //validate();
    //pack();

    if(iterations >= ThreeBGlobalConstants.critical_iterations)
    {
        exportButton.setBackground(exportButtonColor);
    }
    else
    {
        exportButton.setBackground(Color.RED);
    }

}
```

10.6.3.10 void EControlPanel::windowOpened (WindowEvent *e*) [inline]

Definition at line 1123 of file three_B.java.

```
{ }
```

10.6.3.11 void EControlPanel::windowClosed (WindowEvent *e*) [inline]

Definition at line 1124 of file three_B.java.

```
{}
```

10.6.3.12 void EControlPanel::windowDeactivated (WindowEvent *e*) [inline]

Definition at line 1125 of file three_B.java.

```
{ }
```

10.6.3.13 void EControlPanel::windowActivated (WindowEvent *e*) [inline]

Definition at line 1126 of file three_B.java.

```
{ }
```

10.6.3.14 void EControlPanel::windowDeiconified (WindowEvent *e*) [inline]

Definition at line 1127 of file three_B.java.

```
{ }
```

10.6.3.15 void EControlPanel::windowIconified (WindowEvent *e*) [inline]

Definition at line 1128 of file three_B.java.

```
{ }
```

10.6.3.16 void EControlPanel::windowClosing (WindowEvent *e*) [inline]

Send a stop message to the thread if the window is closed.

Definition at line 1131 of file three_B.java.

References ThreeBRunner::has_stopped(), ThreeBRunner::stop_thread(), and tbr.

```
    {
        if(tbr != null && !tbr.has_stopped())
        {
```

```
        int confirmed = JOptionPane.showConfirmDialog(null, "3B still running
  ! Closing will terminate the run. Really close?", "User Confirmation", JOptionPan
  e.YES_NO_OPTION);

        if (confirmed == JOptionPane.YES_OPTION)
        {
            dispose();
            tbr.stop_thread();
        }

    }
    else
    {
        //OK to close
        dispose();
    }
}
```

10.6.3.17 void EControlPanel::export_reconstruction_as_ij ( ) [inline, package]

Generate a fresh ImageJ image in a standard imageJ window, so that ti can be manipulated using the usual ImageJ commands.

The image is size calibrated, so scalebars are easy to add.

Definition at line 1154 of file three_B.java.

References filename, FloatSliderWithBox::getValue(), linear_reconstruction, and reconstructed_pixel_size.

Referenced by ExportButtonListener::actionPerformed().

```
{
    ImageProcessor export = linear_reconstruction.getProcessor().duplicate();

    ImagePlus export_win = new ImagePlus(filename + " reconstruction", export
  );
    export_win.getCalibration().pixelWidth = reconstructed_pixel_size.
  getValue();
    export_win.getCalibration().pixelHeight = reconstructed_pixel_size.
  getValue();
    export_win.getCalibration().setXUnit("nm");
    export_win.getCalibration().setYUnit("nm");
    export_win.show();
    export_win.updateAndDraw();
}
```

10.6.3.18 synchronized FloatProcessor EControlPanel::reconstruct ( ) [inline, private]

Compute a brand new reconstruction.

Definition at line 1167 of file three_B.java.

References pixel_size_in_nm, pts, reconstruction_blur_fwhm, roi, and zoom.

Referenced by EControlPanel(), and update_canvas().

```
{
    //Reconstruct an image which is based around the ROI in size
    int xoff = (int)roi.x;
    int yoff = (int)roi.y;

    int xsize = (int)Math.ceil(roi.width * zoom);
    int ysize = (int)Math.ceil(roi.height * zoom);

    //New blank image set to zero
    FloatProcessor reconstructed = new FloatProcessor(xsize, ysize);

    for(int i=0; i < pts.size(); i++)
    {

        //Increment the count
        int xc = (int)Math.floor((pts.get(i).x-xoff) * zoom + 0.5);
        int yc = (int)Math.floor((pts.get(i).y-yoff) * zoom + 0.5);
        float p = reconstructed.getPixelValue(xc, yc);
        reconstructed.putPixelValue(xc, yc, p+1);
    }

    double blur_sigma = reconstruction_blur_fwhm / (2 * Math.sqrt(2 * Math.lo
  g(2))) * zoom / pixel_size_in_nm;

    (new GaussianBlur()).blurGaussian(reconstructed, blur_sigma, blur_sigma,
  0.005);
    return reconstructed;
}
```

10.6.3.19 void EControlPanel::issue_stop ( ) [inline]

Definition at line 1196 of file three_B.java.

References ThreeBRunner::stop_thread(), stopButton, and tbr.

Referenced by StopButtonListener::actionPerformed().

```
{
    if(tbr != null)
        tbr.stop_thread();
    stopButton.setEnabled(false);
}
```

10.6.3.20 synchronized void EControlPanel::append (final ArrayList< Spot > *p*, int *its*) [inline]

Callback issued by the runnre thread which appends the latest set of points to the array.

This is synchronized since it involves the shared writable array of points.

Definition at line 1208 of file three_B.java.

References iterations, and pts.

Referenced by ThreeBLoader::run(), and ThreeBRunner::send_new_points().

```
{
    for(int i=0; i < p.size(); i++)
        pts.add(p.get(i));

    iterations = its;
}
```

10.6.3.21 void EControlPanel::send_update_canvas_event ( ) [inline]

Callback for causing a recomputation of the reconstruction.

This is a safe method for telling the class to recompute and redraw the reconstruction. It can be called from anywhere, but ensures that the redraw happens in the GUI thread.

Definition at line 1220 of file three_B.java.

References update_canvas().

Referenced by ThreeBLoader::run(), ThreeBRunner::send_new_points(), and SomethingChanges::stateChanged().

```
{
    SwingUtilities.invokeLater(
        new Runnable() {
            final public void run() {
                    update_canvas();
                }
            }
    );
}
```

10.6.3.22 void EControlPanel::die (final String *s*) [inline]

Callback which causes a fatal termination of the 3B control panel.

This is generally caused by something changing such that the error condition was not seen in the Java code, but was seen in C++. Ought never to be called.

Definition at line 1235 of file three_B.java.

Referenced by ThreeBRunner::die().

```
{
    SwingUtilities.invokeLater(
        new Runnable() {
            final public void run() {
                    ij.IJ.showStatus("3B run terminated due to an error.\n");

                    JOptionPane.showMessageDialog(null, "Error: "+s, "Fatal e
  rror", JOptionPane.ERROR_MESSAGE);
                    dispose();
                }
```

```
            }
        );
    }
```

10.6.3.23 void EControlPanel::send_status_text_message (final String *s*) [inline]

Callback to update the status message safely.

Definition at line 1249 of file three_B.java.

References set_status().

Referenced by ThreeBLoader::run(), ThreeBRunner::run(), and ThreeBRunner::send_message_string().

```
    {
        SwingUtilities.invokeLater(
            new Runnable(){
                final public void run() {
                        set_status(s);
                    }
                }
        );
    }
```

10.6.3.24 void EControlPanel::send_time_text_message (final String *s*) [inline]

Callback to update the status message safely.

Definition at line 1262 of file three_B.java.

References set_time().

Referenced by ThreeBRunner::send_new_points().

```
    {
        SwingUtilities.invokeLater(
            new Runnable(){
                final public void run() {
                        set_time(s);
                    }
                }
        );
    }
```

10.6.4 Member Data Documentation

10.6.4.1 ImagePlus EControlPanel::linear_reconstruction [private]

Definition at line 890 of file three_B.java.

Referenced by EControlPanel(), export_reconstruction_as_ij(), and update_canvas().

10.6.4.2 ImageCanvas EControlPanel::canvas [private]

Definition at line 891 of file three_B.java.

Referenced by EControlPanel(), and update_canvas().

10.6.4.3 JButton EControlPanel::stopButton [private]

Definition at line 892 of file three_B.java.

Referenced by EControlPanel(), and issue_stop().

10.6.4.4 JButton EControlPanel::exportButton [package]

Definition at line 892 of file three_B.java.

Referenced by EControlPanel(), and update_canvas().

10.6.4.5 JButton EControlPanel::closeButton [package]

Definition at line 892 of file three_B.java.

Referenced by EControlPanel().

10.6.4.6 Color EControlPanel::exportButtonColor [private]

Definition at line 893 of file three_B.java.

Referenced by EControlPanel(), and update_canvas().

10.6.4.7 JLabel EControlPanel::status [private]

Definition at line 894 of file three_B.java.

Referenced by EControlPanel(), and set_status().

10.6.4.8 JLabel EControlPanel::time_msg [package]

Definition at line 894 of file three_B.java.

Referenced by EControlPanel(), and set_time().

10.6.4.9 FloatSliderWithBox EControlPanel::blur_fwhm [private]

Definition at line 895 of file three_B.java.

Referenced by EControlPanel(), and update_canvas().

10.6.4.10 FloatSliderWithBox EControlPanel::reconstructed_pixel_size [private]

Definition at line 896 of file three_B.java.

Referenced by EControlPanel(), export_reconstruction_as_ij(), and update_canvas().

10.6.4.11 Panel EControlPanel::complete [private]

Definition at line 898 of file three_B.java.

Referenced by EControlPanel().

10.6.4.12 JPanel EControlPanel::buttons [private]

Definition at line 899 of file three_B.java.

Referenced by EControlPanel().

10.6.4.13 ThreeBRunner EControlPanel::tbr [private]

Definition at line 901 of file three_B.java.

Referenced by EControlPanel(), issue_stop(), and windowClosing().

10.6.4.14 ArrayList<Spot> EControlPanel::pts [private]

Definition at line 902 of file three_B.java.

Referenced by append(), EControlPanel(), and reconstruct().

10.6.4.15 Rectangle EControlPanel::roi [private]

Definition at line 905 of file three_B.java.

Referenced by EControlPanel(), and reconstruct().

10.6.4.16 double EControlPanel::zoom = 0.01 [private]

Definition at line 906 of file three_B.java.

Referenced by reconstruct(), and update_canvas().

10.6.4.17 double EControlPanel::reconstruction_blur_fwhm = .5 [private]

Definition at line 908 of file three_B.java.

Referenced by reconstruct(), and update_canvas().

10.6.4.18 double EControlPanel::pixel_size_in_nm [private]

Definition at line 910 of file three_B.java.

Referenced by EControlPanel(), reconstruct(), and update_canvas().

10.6.4.19 String EControlPanel::filename [private]

Definition at line 911 of file three_B.java.

Referenced by EControlPanel(), and export_reconstruction_as_ij().

10.6.4.20 int EControlPanel::iterations = 0 [private]

Definition at line 915 of file three_B.java.

Referenced by append(), and update_canvas().

The documentation for this class was generated from the following file:

- three_B.java

10.7 ExportButtonListener Class Reference

Listener for the export button.

Public Member Functions

- ExportButtonListener (EControlPanel c_)
- void actionPerformed (ActionEvent e)

Private Attributes

- EControlPanel c

10.7.1 Detailed Description

Listener for the export button.

Definition at line 872 of file three_B.java.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 ExportButtonListener::ExportButtonListener (EControlPanel *c_*) [inline]

Definition at line 875 of file three_B.java.

References c.

```
{
    c = c_;
}
```

10.7.3 Member Function Documentation

10.7.3.1 void ExportButtonListener::actionPerformed (ActionEvent *e*) [inline]

Definition at line 880 of file three_B.java.

References c, and EControlPanel::export_reconstruction_as_ij().

```
{
    c.export_reconstruction_as_ij();
}
```

10.7.4 Member Data Documentation

10.7.4.1 EControlPanel ExportButtonListener::c [private]

Definition at line 874 of file three_B.java.

Referenced by actionPerformed(), and ExportButtonListener().

The documentation for this class was generated from the following file:

- three_B.java

10.8 FitSpots Class Reference

Mega class which actually does the meat of the spot fitting.

Public Member Functions

- FitSpots (const vector< Image< float > > &ims_, FitSpotsGraphics &graphics_, UserInterfaceCallback &ui_, StateParameters &params, ofstream &save_spots_)
- void optimize_each_spot_in_turn_for_several_passes ()
- void try_modifying_model ()
- void run ()

Private Attributes

- const vector< Image< float > > & ims
- FitSpotsGraphics & graphics
- UserInterfaceCallback & ui
- const vector< ImageRef > pixels
- vector< Vector< 4 > > spots
- const int start_iteration
- int start_pass
- MT19937 & rng
- const double variance
- const double intensity_mu
- const double intensity_sigma
- const double blur_mu
- const double blur_sigma
- const double area_extra_radius
- set< ImageRef > allowed_area
- const int use_position_prior
- const double position_prior
- const double max_motion
- const int sample_iterations
- const int main_cg_max_iterations
- const int main_samples
- const int main_passes
- const int outer_loop_iterations
- const int add_remove_tries
- const int add_remove_opt_samples
- const int add_remove_opt_retries
- const int add_remove_opt_hess_inner_samples
- const int h_outer_samples
- const int h_inner_samples

- const int tsamples
- const Image< float > ave
- ofstream & save_spots
- double time_gibbs
- double time_cg
- const bool scale_brightness_limit
- const Vector< 4 > limit
- const Matrix< 3 > A
- const Vector< 3 > pi
- vector< vector< double > > pixel_intensities
- DataForMCMC data_for_t_mcmc
- DataForMCMC data_for_h_mcmc
- int iteration

10.8.1 Detailed Description

Mega class which actually does the meat of the spot fitting.

probably could be refactored a bit.

Definition at line 1348 of file multispot5.cc.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 FitSpots::FitSpots (const vector< Image< float > > & *ims_*, FitSpotsGraphics & *graphics_*, UserInterfaceCallback & *ui_*, StateParameters & *params*, ofstream & *save_spots_*) [inline]

Definition at line 1438 of file multispot5.cc.

References assert_same_size().

```
:ims(ims_), graphics(graphics_),ui(ui_),

 //Set up the main parameters
 pixels(params.pixels),
 spots(params.spots),
 start_iteration(params.iteration),
 start_pass(params.pass),
 rng(*params.rng),



 //Set up all the system parameters
 variance(1), // it should be

 //To scale the X axis of a log-normal distribution, only
 //the mu parameter needs to be changed...
 intensity_mu(GV3::get<double>("intensity.rel_mu", 0., -1)  + log(sqrt(
  variance))),
 intensity_sigma(GV3::get<double>("intensity.rel_sigma", 0., -1)),
```

```
    blur_mu(GV3::get<double>("blur.mu", 0., -1)),
    blur_sigma(GV3::get<double>("blur.sigma", 0., -1)),

    //Spot position prior
    area_extra_radius(GV3::get<double>("position.extra_radius", 0., -1)),
    allowed_area(dilate_mask(pixels, area_extra_radius)),
    use_position_prior(GV3::get<bool>("position.use_prior", true, -1)),
    position_prior(1.0 / allowed_area.size()),

    //General optimizing and sampling parameters
    max_motion(GV3::get<double>("cg.max_motion", 0., -1)),
    sample_iterations(GV3::get<int>("gibbs.mixing_iterations", 0, -1)),

    //Task specific optimizing and sampling parameters:

    //Main optimization loop
    main_cg_max_iterations(GV3::get<double>("main.cg.max_iterations", 0., -1)),
    main_samples(GV3::get<int>("main.gibbs.samples", 0, -1)),
    main_passes(GV3::get<int>("main.passes", 0, -1)),
    outer_loop_iterations(GV3::get<int>("main.total_iterations", 100000000, 1)),


    //Spot selection loop
    add_remove_tries(GV3::get<int>("add_remove.tries", 0, -1)),
    add_remove_opt_samples(GV3::get<int>("add_remove.optimizer.samples", 0, -1))
     ,
    add_remove_opt_retries(GV3::get<int>("add_remove.optimizer.attempts", 0, -1)
     ),
    add_remove_opt_hess_inner_samples(GV3::get<int>("add_remove.optimizer.hessia
     n_inner_samples", 0, -1)),
    h_outer_samples(GV3::get<int>("add_remove.hessian.outer_samples", 0, -1)),
    h_inner_samples(GV3::get<int>("add_remove.hessian.inner_samples", 0, -1)),
    tsamples(GV3::get<int>("add_remove.thermo.samples", 0, -1)),

    ave(average_image(ims_)),

    save_spots(save_spots_),

    time_gibbs(0),
    time_cg(0),

    scale_brightness_limit(GV3::get<bool>("max_motion.use_brightness_std", 0, -1
     )),
    limit(makeVector(brightness_motion_limit(intensity_mu, intensity_sigma,
     scale_brightness_limit), 1, 1, 1)*max_motion),

    A(GV3::get<Matrix<3> >("A", Zeros, 1)),
    pi(GV3::get<Vector<3> >("pi", Zeros, 1)),


    data_for_t_mcmc(pixels, pixel_intensities, intensity_mu, intensity_sigma,
     blur_mu, blur_sigma, variance, tsamples, sample_iterations, A, pi, rng),
    data_for_h_mcmc(pixels, pixel_intensities, intensity_mu, intensity_sigma,
     blur_mu, blur_sigma, variance, h_outer_samples, sample_iterations, A, pi, rng)

{
    assert_same_size(ims);

    //Pixel intensities for all images [frame][pixel]
    pixel_intensities.resize(ims.size(), vector<double>(pixels.size()));
    for(unsigned int frame=0; frame < ims.size(); frame++)
        for(unsigned int p=0; p < pixels.size(); p++)
```

```
            pixel_intensities[frame][p] = ims[frame][pixels[p]];
}
```

10.8.3 Member Function Documentation

10.8.3.1 void FitSpots::optimize_each_spot_in_turn_for_several_passes ( ) [inline]

Perform a complete iteration of the optimzation stage of the spot firrint algorithm.

Definition at line 1516 of file multispot5.cc.

References boundingbox(), SampledMultispot::compute_spot_intensity(), get_spot_pixels(), FreeEnergyHessian::hessian(), ConjugateGradientOnly< Size, Precision >::iterate(), ConjugateGradientOnly< Size, Precision >::max_iterations, SampledMultispot::GibbsSampler2::next(), SampledMultispot::remove_spot(), SampledMultispot::GibbsSampler2::sample(), SampledMultispot::GibbsSampler2::sa intensities(), sampled_background_spot_hessian_ffbs(), scale_to_bytes(), SampledMultispot::sequence(), spots_to_Vector(), TIME, and ConjugateGradientOnly< Size, Precision >::x.

```
{
    //Precompute the intensities for all spot pixels
    vector<vector<double> > spot_intensities; //[spot][pixel]
    for(unsigned int i=0; i < spots.size(); i++)
        spot_intensities.push_back(compute_spot_intensity(pixels, spots[i]));


    //Which pixels does each spot have?
    vector<vector<int> > spot_pixels; //[spot][pixel]
    spot_pixels.resize(spots.size());
    for(unsigned int s=0; s < spots.size(); s++)
        get_spot_pixels(pixels, spots[s], spot_pixels[s]);


    //Optimize the model, N spots at a time.
    //
    for(int pass=start_pass; pass < main_passes; pass++)
    {
        save_spots << "Pass: " << pass << endl;
        rng.write(save_spots);
        save_spots << endl;

        start_pass=0; // This is only nonzero first time, since we might chek
  point midway through an iteration
        save_spots << "PASS" << pass << ": " << setprecision(20) << scientifi
  c << spots_to_Vector(spots) << endl;
        save_spots << "ENDCHECKPOINT" << endl << flush;

        ui.per_pass(iteration, pass, spots);
        //Sort spots according to pass%4

        //Sort the spots so that the optimization runs in sweeps
        //This heiristic seems to increase the rate of propagation of informa
  tion
        //about spot positions.
```

```
      //Create a list of indices
      vector<int> index = sequence(spots.size());

      int passs = pass + iteration;
      //Sort the indices according to the position of the spot that they in
dex
      if(passs%4 == 0)
          sort(index.begin(), index.end(),
IndexLexicographicPosition<less<double>, 2>(spots));
      else if(passs%4==1)
          sort(index.begin(), index.end(),
IndexLexicographicPosition<greater<double>, 2>(spots));
      else if(passs%4==1)
          sort(index.begin(), index.end(),
IndexLexicographicPosition<less<double>, 3>(spots));
      else
          sort(index.begin(), index.end(),
IndexLexicographicPosition<greater<double>, 3>(spots));

      //Reorder the spots and their intensities and their pixel lists
      {
          vector<Vector<4> > tmp_spot(index.size());
          vector<vector<double> > tmp_spot_intensities(index.size());
          vector<vector<int> > tmp_spot_pixels(index.size());
          for(unsigned int i=0; i < index.size(); i++)
          {
              tmp_spot[i] = spots[index[i]];
              swap(tmp_spot_intensities[i], spot_intensities[index[i]]);
              swap(tmp_spot_pixels[i], spot_pixels[i]);
          }

          swap(tmp_spot, spots);
          swap(tmp_spot_intensities, spot_intensities);
          swap(tmp_spot_pixels, spot_pixels);
      }

      //Sweep through and optimize each spot in turn
      for(int s=0; s < (int)spots.size(); s++)
      {
          ui.per_spot(iteration, pass, s, spots.size());
          ui.perhaps_stop();

          TIME(timer.reset();)
          //Get some samples with Gibbs sampling
          vector<vector<vector<State> > > sample_list; //[N][spot][frame]:
list of samples drawn using Gibbs sampling
          vector<vector<vector<double> > > sample_intensities; //[sample][f
rame][pixel]

          GibbsSampler2 sampler(pixel_intensities, spot_intensities, spots,
 spot_pixels, A, pi, variance, sample_iterations);
          for(int i=0; i < main_samples; i++)
          {
              sampler.next(rng);
              sample_list.push_back(sampler.sample());
              sample_intensities.push_back(sampler.sample_intensities());

              ui.perhaps_stop();
          }

          //First, remove the spot from all the samples.
          for(unsigned int i=0; i < sample_list.size(); i++)
```

```
            remove_spot(sample_intensities[i], spot_intensities[s], sampl
e_list[i][s]);

          //cout << timer.get_time() << endl;
          TIME(time_gibbs += timer.reset();)

          //Package up all the data
          SampledBackgroundData data(sample_intensities, pixel_intensities,
 pixels,
                                     intensity_mu, intensity_sigma,
blur_mu, blur_sigma,
                                     A, pi, variance);

          //Derivative computer:
          SpotNegProbabilityDiffWithSampledBackground compute_deriv(data);


          graphics.draw_pixels(pixels, 0, 0, 1, 1);
          graphics.draw_krap(spots, scale_to_bytes(ave), boundingbox(
pixels), s);
          graphics.swap();

          //Optimize spot "s"
          //Optimize with the derivatives only since the actual probability

          //is much harder to compute
          ConjugateGradientOnly<4> cg(spots[s], compute_deriv, limit);


          cg.max_iterations = main_cg_max_iterations;


          #if 0
              cout << setprecision(10);
              cout << spots_to_Vector(spots) << endl;
              Matrix<4> hess, hess_errors;
              cout << "Hello, my name is Inigo Montoya\n";
              /*for(int i=0; i < 4; i++)
              {
                  Matrix<4, 2> d = numerical_gradient_with_errors(NthDeriv(
compute_deriv, i), cg.x);
                  hess[i] = d.T()[0];
                  hess_errors[i] = d.T()[1];
              }
              */
              //cout << "Errors:\n" << hess_errors << endl;
              //cout << "NHess:\n" << hess<< endl;
              Matrix<4> rhess =  -sampled_background_spot_hessian(cg.x, dat
a);
              cout << "Hess:\n" << rhess << endl;
              //cout << "Err:\n" << hess - rhess << endl;

              //Vector<4> grad = compute_deriv(cg.x);

              //Matrix<4> e = hess - rhess;

              //for(int i=0; i < 4; i++)
              //  for(int j=0; j < 4; j++)
              //      e[i][j] /= hess_errors[i][j];

              //cout << "Err:\n" << e << endl;
              cout << "Deriv:" <<  compute_deriv(cg.x) << endl;
```

```
            //cout << "Full:\n" << sampled_background_spot_hessian2(cg.x,
 data) - grad.as_col()*grad.as_row() << endl;

            FreeEnergyHessian hesscomputer(data_for_h_mcmc);

            Matrix<4> nhess = hesscomputer.hessian(spots, 0);
            cout << "NHess:\n" << nhess << endl;

            cout << "Turbo-N Hess:\n" <<
sampled_background_spot_hessian_ffbs(cg.x, data, 10000) << endl;

            cout << "TI energy: " << NegativeFreeEnergy(data_for_t_mcmc)(
spots_to_Vector(spots)) << endl;
            cout << "FA energy: " <<  SpotProbabilityWithSampledBackgroun
dFAKE(data)(cg.x) << endl;



            //cout << "Numerical hessian from FD:\n" << numerical_hessian
(SpotProbabilityWithSampledBackgroundFAKE(data), cg.x) << endl;
            exit(0);
        #endif
        //cout << "Starting opt... " << cg.x << endl;
        while(cg.iterate(compute_deriv))
        {
            graphics.draw_krap(spots, scale_to_bytes(ave), boundingbox(
pixels), s, cg.x);
            graphics.draw_pixels(pixels, 0, 0, 1, .2);
            graphics.swap();
            ui.perhaps_stop();
            //cout << cg.x << endl;
        }

        //Update to use the result of the optimization
        spots[s] = cg.x;
        //cout << "End: " << cg.x << endl;

        graphics.draw_krap(spots, scale_to_bytes(ave), boundingbox(
pixels), -1);
        graphics.swap();

        //Recompute the new spot intensity, since the spot has changed
        spot_intensities[s] = compute_spot_intensity(pixels, spots[s]);

        //Recompute which are the useful pixels
        get_spot_pixels(pixels, spots[s], spot_pixels[s]);

        //Is the spot within the allowed area, i.e. is it's prior 0?
        //The prior is sero only if it we are using it and we're in an in
valid area

        ImageRef quantized_spot_position = ir_rounded(spots[s].slice<2,2>
());
        bool zero_prior = use_position_prior && (allowed_area.count(quant
ized_spot_position)==0);

        //Check to see if spot has been ejected. If spot_pixels is empty,
 then it has certainly been ejected.
        if(spot_pixels[s].empty() || zero_prior)
        {
            //Spot has been ejected. Erase it.
            cout  << " Erasing ejected spot: " << spot_pixels[s].empty()
```

```
  << " " << zero_prior << endl;
                cout << spots[s] << endl;
                //GUI_Pause(1);

                spot_intensities.erase(spot_intensities.begin() + s);
                spot_pixels.erase(spot_pixels.begin() + s);
                spots.erase(spots.begin() + s);
                s--;
                //exit(0);
            }

            //cout << timer.get_time() << endl;
            TIME(time_cg += timer.reset();)

            //cout << "Times: " << time_gibbs << " " << time_cg << endl;
            //save_spots << "INTERMEDIATE: " << setprecision(15) << scientifi
  c << spots_to_Vector(spots) << endl;
        }
    }
}
```

10.8.3.2 void FitSpots::try_modifying_model ( ) [inline]

Perform a complete iteration of the model size modification stage of the spot fitting algorithm.

Definition at line 1729 of file multispot5.cc.

References SampledMultispot::add_spot(), boundingbox(), SampledMultispot::compute_spot_intensity(), NegativeFreeEnergy::compute_with_mask(), get_spot_pixels(), ConjugateGradientOnly< Size, Precision >::init(), ConjugateGradientOnly< Size, Precision >::iterate(), ln(), log_normal_mode(), SampledMultispot::GibbsSampler::next(), SampledMultispot::GibbsSampler2::next(), SampledMultispot::remove_spot(), SampledMultispot::GibbsSampler::sample(), SampledMultispot::GibbsSampler2::sample(), SampledMultispot::GibbsSampler::sample_intensities(), SampledMultispot::GibbsSampler2::sample_intensities(), sampled_background_spot_hessian_ffbs(), scale_to_bytes(), spots_to_Vector(), and ConjugateGradientOnly< Size, Precision >::x.

```
{

    for(int i=0; i < add_remove_tries; i++)
    {
        ui.per_modification(iteration, i, add_remove_tries);
        ui.perhaps_stop();

        cout << endl << endl << "Modifying the model" << endl << "===========
  ==========\n";
        cout << "Hello\n";
        bool add_spot = (rng() > 0.5) || (spots.size() == 1);
        cout << "World\n";

        vector<Vector<4> > model_1, model_2;


        int original; //What is the original model? Model 1 or Model 2?

        if(add_spot)
```

```
      {
          model_1 = spots;
          model_2 = model_1;

          //Pick a pixel within the thresholded ones as a starting point
          int r;
          do
          {
              r = (int)(rng() * pixels.size());
              //cout << r << " " << log_ratios[pixels[r]] << " " << pixels[
r] << " " << threshold << endl;
          }
          while(0);

          //This version does not (yet?) suppotrt thresholding on log_ratio
s
          //for placing spots, since the purpose of log_ratios has diminish
ed.
          //while(log_ratios[pixels[r]] < threshold);


          model_2.push_back(makeVector(log_normal_mode(intensity_mu,
intensity_sigma),
                                       log_normal_mode(blur_mu, blur_sigma)
,
                                       pixels[r].x + rng()-.5, pixels[r].y
+ rng() - .5));
          cout << "Adding a spot\n";

          original = 1;
      }
      else
      {
          //Pick a random point to optimize/remove
          int a_random_spot = static_cast<int>(rng() * spots.size());
          model_1 = spots;
          swap(model_1[model_1.size()-1], model_1[a_random_spot]);

          model_2 = model_1;

          model_1.pop_back();
          cout << "Removing a spot\n";
          original = 2;
      }

      //The mobile spot is always the last spot of model_2
      const int spot = model_2.size() - 1;

      cout << "Original model: " << original << endl;

      //Precompute the intensities for all spot pixels
      //Model 2 is always one longer than model 1 and
      //differs only on the extra element
      vector<vector<double> > model2_spot_intensities; //[spot][pixel]
      for(unsigned int i=0; i < model_2.size(); i++)
          model2_spot_intensities.push_back(compute_spot_intensity(pixels,
model_2[i]));

      //Which pixels does each spot have?
      vector<vector<int> > model2_spot_pixels(model_2.size()); //[spot][pix
el]
      for(unsigned int s=0; s < model_2.size(); s++)
```

```
            get_spot_pixels(pixels, model_2[s], model2_spot_pixels[s]);

        //Optimize spot:
        {
            cout << "Optimizing spot for model selection\n";


            //Get some samples with Gibbs sampling
            vector<vector<vector<State> > > sample_list; //[N][spot][frame]:
list of samples drawn using Gibbs sampling
            vector<vector<vector<double> > > sample_intensities; //[sample][f
rame][pixel]

            GibbsSampler2 sampler(pixel_intensities, model2_spot_intensities,
 model_2, model2_spot_pixels, A, pi, variance, sample_iterations);
            for(int i=0; i < add_remove_opt_samples; i++)
            {
                sampler.next(rng);
                sample_list.push_back(sampler.sample());
                sample_intensities.push_back(sampler.sample_intensities());
                ui.perhaps_stop();
            }

            //First, remove the spot from all the samples.
            for(unsigned int i=0; i < sample_list.size(); i++)
                remove_spot(sample_intensities[i], model2_spot_intensities[sp
ot], sample_list[i][spot]);

            //Package up all the data
            SampledBackgroundData data(sample_intensities, pixel_intensities,
 pixels,
                                       intensity_mu, intensity_sigma,
blur_mu, blur_sigma,
                                       A, pi, variance);

            //Derivative computer:
            SpotNegProbabilityDiffWithSampledBackground compute_deriv(data);

            graphics.draw_krap(model_2, scale_to_bytes(ave), boundingbox(
pixels), spot);
            graphics.swap();

            //Optimize spot "s"
            //Optimize with the derivatives only since the actual probability

            //is much harder to compute
            ConjugateGradientOnly<4> cg(model_2[spot], compute_deriv, limit);



            for(int attempt=0; attempt < add_remove_opt_retries; attempt++)
            {
                cout << "Attempt " << attempt << " of " << add_remove_opt_ret
ries << endl;
                ui.perhaps_stop();

                //Optimize with conjugate gradient
                while(cg.iterate(compute_deriv))
                {
                    ui.perhaps_stop();
                    graphics.draw_krap(model_2, scale_to_bytes(ave),
boundingbox(pixels), spot, cg.x);
```

```
                graphics.swap();
            }

            //Check for being at a saddle point (no point checking on the
 last try)
            //All eigenvectors should be negative at a maximum.
            //WTF: is this a bug? WTF?
            //It was this:
            //if(attempt < add_remove_tries - 1)
            if(attempt < add_remove_opt_retries - 1)
            {
                Matrix<4> hessian = sampled_background_spot_hessian_ffbs(
cg.x, data, add_remove_opt_hess_inner_samples, rng);
                SymEigen<4> hess_decomp(hessian);

                //cout << "What the fuck:" << endl << hessian << endl <<
hessian3<< endl << hessian2 << endl;

                cout << "Eigenvalues are: " << hess_decomp.get_evalues()
<< endl;

                if(hess_decomp.is_negdef())
                    break;
                else
                {
                    //Restart in the direction of the best uphill part
                    cg.init(cg.x + 0.1 * hess_decomp.get_evectors()[3], (
hess_decomp.get_evectors()[3]));


                    cout << "Grad = " << compute_deriv(cg.x) << endl;
                    for(int i=0; i < 4; i++)
                    {
                        cout << "Direction: " << i << endl;
                        cout << unit(compute_deriv(cg.x + 0.1*hess_decomp
.get_evectors()[i])) * hess_decomp.get_evectors()[i] << endl;
                    }

                    for(int i=0; i < 4; i++)
                    {
                        cout << "Direction: " << i << endl;
                        Vector<4> d = Zeros;
                        d[i] = 1;
                        cout << unit(compute_deriv(cg.x + d)) * hess_deco
mp.get_evectors()[i] << endl;
                    }
                }
            }
        }


        //Update to use the result of the optimization
        model_2[spot] = cg.x;

        graphics.draw_krap(model_2, scale_to_bytes(ave), boundingbox(
pixels), -1);
        graphics.swap();


        //Update cached data based on the new spot position
        model2_spot_intensities[spot] = compute_spot_intensity(pixels, mo
del_2[spot]);
```

```
            get_spot_pixels(pixels, model_2[spot], model2_spot_pixels[spot]);


            cout << "Done optimizing for model selection\n";
        }


        //Which model to keep?
        int keep=original;

        //Compute position prior (and we might be able to reject it really qu
ickly here)
        bool zero_prior = use_position_prior && (allowed_area.count(ir_rounde
d(model_2[spot].slice<2,2>()))==0);

        if(zero_prior)
        {
            //Model 2 went bad, so we clearly keep model 1
            keep = 1;
        }
        else
        {

            //The position prior is independent
            //Compute the difference  model2 - model1
            //This is only valid if model2 is in the valid region
            double position_log_prior_model2_minus_model1;
            if(use_position_prior)
                position_log_prior_model2_minus_model1 = (model_2.size() - mo
del_1.size()) * ln(position_prior);
            else
                position_log_prior_model2_minus_model1 = 0;


            //Integrate model_2
            //First compute the Hessian since this might go wrong.

            //FreeEnergyHessian hesscomputer(data_for_h_mcmc);
            Matrix<4> hess;// = hesscomputer.hessian(model_2, spot);

            //Use turbohess here since it is much faster, as the backwards sa
mpling step is fast
            //We expect this hessian to be really quite different, if the spo
t has moved from
            //a long way away, since the sampling will have changed dramatica
lly
            {
                //Get some samples with Gibbs sampling
                vector<vector<vector<State> > > sample_list; //[N][spot][fram
e]: list of samples drawn using Gibbs sampling
                vector<vector<vector<double> > > sample_intensities; //[sampl
e][frame][pixel]

                GibbsSampler sampler(pixel_intensities, model2_spot_intensiti
es, model_2, A, pi, variance, sample_iterations);
                for(int i=0; i < h_outer_samples; i++)
                {
                    ui.perhaps_stop();
                    sampler.next(rng);
                    sample_list.push_back(sampler.sample());
                    sample_intensities.push_back(sampler.sample_intensities()
);
```

```
            }

            //First, remove the spot from all the samples.
            for(unsigned int i=0; i < sample_list.size(); i++)
                remove_spot(sample_intensities[i], model2_spot_intensitie
s[spot], sample_list[i][spot]);

            //Package up all the data
            SampledBackgroundData data(sample_intensities,
pixel_intensities, pixels,
                                       intensity_mu, intensity_sigma,
blur_mu, blur_sigma,
                                       A, pi, variance);

            hess = sampled_background_spot_hessian_ffbs(model_2[spot], da
ta, h_inner_samples, rng);
        }


        double det = determinant(-hess / (M_PI*2));
        SymEigen<4> hess_decomp(-hess);
        cout << "Hessien Eigenvalues are: " << hess_decomp.get_evalues()
<< endl;
        const double smallest_evalue = 1e-6;

        //If the determinant is negative, then we are still at a saddle p
oint
        //despite the attempts above, so abandon the attempt.
        if(hess_decomp.get_evalues()[0] >  smallest_evalue)
        {

            //Compute the free energy of model 2 at the MLE estimate
            cout << "Model 2:\n";
    //      double model_2_energy = -NegativeFreeEnergy(data_for_t_mcmc)(
spots_to_Vector(model_2));
            double model_2_energy = -NegativeFreeEnergy(data_for_t_mcmc).
compute_with_mask(spots_to_Vector(model_2), model2_spot_pixels);
            cout << "Energy: " << model_2_energy << endl;

            //Combine the MLE energy and Hessian using Laplace's approxim
ation
            double model_2_occam =  -0.5 * log(det);
            double model_2_prob = model_2_energy + model_2_occam + positi
on_log_prior_model2_minus_model1;

            cout << "Occam: " << model_2_occam << endl;
            cout << "Position: " << position_log_prior_model2_minus_model
1 << endl;
            cout << "Prob: " << model_2_prob << endl;

            //Integrate model_1
            //It has no parameters, in this formulation.
            //double model_1_prob = -NegativeFreeEnergy(data_for_t_mcmc)(
spots_to_Vector(model_1));

            //Note that model_1 always has one fewer spots, and the last
spot is always the one
            //missing, so we can make the correct mask very easily:
            model2_spot_pixels.pop_back();
            double model_1_prob = -NegativeFreeEnergy(data_for_t_mcmc).
compute_with_mask(spots_to_Vector(model_1), model2_spot_pixels);
            cout << "Prob: " << model_1_prob << endl;
```

```
                //model_1_prob = -NegativeFreeEnergy(data_for_t_mcmc)(spots_t
    o_Vector(model_1));

                if(model_2_prob  >  model_1_prob)
                    keep=2;
                else
                    keep=1;

                cout << "Models evaluated\n";
            }
            else
            {
                cout << "Determinant has bad eigenvalues!\n";
                keep = original;
                cout << hess_decomp.get_evalues() << endl;
            }
        }

        if(keep == 2)
        {
            spots = model_2;
            cout << "Keeping model 2\n";

        }
        else
        {
            spots = model_1;
            cout << "Keeping model 1\n";
        }

        if(original != keep)
        {
            cout << "Model changed!\n";
            //break;
        }
    }
}
```

10.8.3.3 void FitSpots::run ( ) [inline]

Run the complete optimization algorithm.

Definition at line 2043 of file multispot5.cc.

References spots_to_Vector().

Referenced by fit_spots_new().

```
{
    graphics.init(ims[0].size());
    save_spots << "LOGVERSION 2" << endl;
    save_spots << "PIXELS";
    for(unsigned int i=0; i < pixels.size(); i++)
        save_spots << " " << pixels[i].x << " " << pixels[i].y;
    save_spots << endl;

    save_spots << "BEGINGVARLIST" << endl;
    GV3::print_var_list(save_spots, "", 1);
    save_spots << "ENDGVARLIST" << endl;
```

```
  //TODO all GVARS are set, so dump out gvars.

  cout << "Limit vector: " << limit << endl;

  for(iteration=start_iteration; iteration < outer_loop_iterations ;
iteration++)
  {
      save_spots << "Iteration: " << iteration << " (" << iteration *
main_passes << ")" << endl;
      save_spots << "MAIN: " << setprecision(20) << scientific <<
spots_to_Vector(spots) << endl;

      cout << endl << endl << "----------------------" << endl << "Optimizi
ng:\n";
      cout << spots.size() << endl;


      optimize_each_spot_in_turn_for_several_passes();

      //spot_intensities is be correct here!
      try_modifying_model();
  }
  save_spots << "FINAL: " << setprecision(15) << scientific <<
spots_to_Vector(spots) << endl;
}
```

10.8.4 Member Data Documentation

10.8.4.1 const vector<Image<float> >& FitSpots::ims [private]

Input data.

Definition at line 1350 of file multispot5.cc.

10.8.4.2 FitSpotsGraphics& FitSpots::graphics [private]

Graphics class.

Definition at line 1351 of file multispot5.cc.

10.8.4.3 UserInterfaceCallback& FitSpots::ui [private]

Callbacks to provide user interface.

Definition at line 1352 of file multispot5.cc.

10.8.4.4 const vector<ImageRef> FitSpots::pixels [private]

Area in which to perform model fitting.

Definition at line 1353 of file multispot5.cc.

10.8.4.5 vector<Vector<4> > FitSpots::spots [private]

State in terms of current spot positions.

Definition at line 1356 of file multispot5.cc.

10.8.4.6 const int FitSpots::start_iteration [private]

Starting iteration number (for restarting from checkpoint)

Definition at line 1359 of file multispot5.cc.

10.8.4.7 int FitSpots::start_pass [private]

Starting pass (for restarting from checkpoint)

Definition at line 1360 of file multispot5.cc.

10.8.4.8 MT19937& FitSpots::rng [private]

Random numbewr generator.

Definition at line 1362 of file multispot5.cc.

10.8.4.9 const double FitSpots::variance [private]

Variance of noise in data. Must be 1.

Definition at line 1364 of file multispot5.cc.

10.8.4.10 const double FitSpots::intensity_mu [private]

Prior for spot intensity.

Definition at line 1365 of file multispot5.cc.

10.8.4.11 const double FitSpots::intensity_sigma [private]

Prior for spot intensity.

Definition at line 1366 of file multispot5.cc.

10.8.4.12 const double FitSpots::blur_mu [private]

Prior for spot shape.

Definition at line 1367 of file multispot5.cc.

10.8.4.13 const double FitSpots::blur_sigma [private]

Prior for spt shape.

Definition at line 1368 of file multispot5.cc.

10.8.4.14 const double FitSpots::area_extra_radius [private]

Extra size beyone marked region in which spots are allowed to exist.

Definition at line 1377 of file multispot5.cc.

10.8.4.15 set<ImageRef> FitSpots::allowed_area [private]

Total allowed region, pixels dilated by area_extra_radius.

Definition at line 1378 of file multispot5.cc.

10.8.4.16 const int FitSpots::use_position_prior [private]

Should a proper prior over position be uesd? A clue: yes.

Definition at line 1379 of file multispot5.cc.

10.8.4.17 const double FitSpots::position_prior [private]

Value for the posision prior, i.e. reciprocal of area.

Definition at line 1380 of file multispot5.cc.

10.8.4.18 const double FitSpots::max_motion [private]

Maximum motion on any axes for optimization. See ConjugateGradientOnly.

Definition at line 1383 of file multispot5.cc.

10.8.4.19 const int FitSpots::sample_iterations [private]

Number of mixing samples to use in Gibbs sampling.

Definition at line 1384 of file multispot5.cc.

10.8.4.20 const int FitSpots::main_cg_max_iterations [private]

Maximum iterations allowed for ConjugateGradientOnly in main optimization loop.

Definition at line 1389 of file multispot5.cc.

10.8.4.21 const int FitSpots::main_samples [private]

Number of samples to use in main loop.

Definition at line 1390 of file multispot5.cc.

10.8.4.22 const int FitSpots::main_passes [private]

Number of passes to perform per iteration in main loop.

Definition at line 1391 of file multispot5.cc.

10.8.4.23 const int FitSpots::outer_loop_iterations [private]

Total number of iterations to perform.

Definition at line 1392 of file multispot5.cc.

10.8.4.24 const int FitSpots::add_remove_tries [private]

Number of attemts to add/remove a spot.

Definition at line 1395 of file multispot5.cc.

10.8.4.25 const int FitSpots::add_remove_opt_samples [private]

Number of samples to use in model modification phase.

Definition at line 1396 of file multispot5.cc.

10.8.4.26 const int FitSpots::add_remove_opt_retries [private]

Number of attempts restarting the optimization to escape saddle points.

Definition at line 1397 of file multispot5.cc.

10.8.4.27 const int FitSpots::add_remove_opt_hess_inner_samples [private]

Number of extra FFBS samples to use for computing Hessian when testing for convergence to non-saddle point.

Definition at line 1398 of file multispot5.cc.

10.8.4.28 const int FitSpots::h_outer_samples [private]

Number of samples used for computing Hessian as part of Laplace's approximation.

Definition at line 1399 of file multispot5.cc.

10.8.4.29 const int FitSpots::h_inner_samples [private]

Number of additional FFBS samples to use for computing Hessian as part of Laplace's approximation.

Definition at line 1400 of file multispot5.cc.

10.8.4.30 const int FitSpots::tsamples [private]

Number of samples to use in thermodynamic integration.

Definition at line 1401 of file multispot5.cc.

10.8.4.31 const Image<float> FitSpots::ave [private]

Average of input data: used for.

Definition at line 1404 of file multispot5.cc.

10.8.4.32 ofstream& FitSpots::save_spots [private]

Output stream for log file.

Definition at line 1408 of file multispot5.cc.

10.8.4.33 double FitSpots::time_gibbs [private]

Benchmarking data.

Definition at line 1411 of file multispot5.cc.

10.8.4.34 double FitSpots::time_cg [private]

Benchmarking data.

Definition at line 1412 of file multispot5.cc.

10.8.4.35 const bool FitSpots::scale_brightness_limit [private]

Motion limit for ConjugateGradientOnly The x distance, y distance and spot size are all approximately the same scale which is of order 1.

The brigntness is not. By default, the brightness limit is also 1. This flag controls whether is should be set to the standard deviation of the brightness prior distribution. This setting will put the motion limit on the same scale as the other three parameters.

Definition at line 1420 of file multispot5.cc.

10.8.4.36 const Vector<4> FitSpots::limit [private]

Limit vector for ConjugateGradientOnly.

Definition at line 1421 of file multispot5.cc.

10.8.4.37 const Matrix<3> FitSpots::A [private]

Transition matrix.

Definition at line 1423 of file multispot5.cc.

10.8.4.38 const Vector<3> FitSpots::pi [private]

Initial probabilities.

Definition at line 1424 of file multispot5.cc.

10.8.4.39 vector<vector<double> > FitSpots::pixel_intensities [private]

Pixel intensities for all images [frame][pixel].

Definition at line 1427 of file multispot5.cc.

10.8.4.40 DataForMCMC FitSpots::data_for_t_mcmc [private]

Aggergated data for thermodynamic integration.

Definition at line 1429 of file multispot5.cc.

10.8.4.41 DataForMCMC FitSpots::data_for_h_mcmc [private]

Aggergated data for finding hessian.

Definition at line 1430 of file multispot5.cc.

10.8.4.42 int FitSpots::iteration [private]

Iteration number.

Definition at line 1432 of file multispot5.cc.

The documentation for this class was generated from the following file:

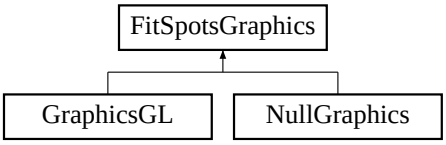
- multispot5.cc

10.9 FitSpotsGraphics Class Reference

Graphics class for FittingSpots.

`#include <multispot5.h>`

Inheritance diagram for FitSpotsGraphics:



Public Member Functions

- virtual void init (CVD::ImageRef size)=0
- virtual void draw_krap (const std::vector< TooN::Vector< 4 > > &spots, const CVD::Image< CVD::byte > &im, const BBox &box, int N, TooN::Vector< 4 > s=TooN::Ones *1e99)=0
- virtual void swap ()=0
- virtual void draw_pixels (const std::vector< CVD::ImageRef > &pix, float r, float g, float b, float a=1)=0
- virtual void draw_bbox (const BBox &bbox)=0
- virtual void glDrawCross (const TooN::Vector< 2 > &p, int size=3)=0
- virtual ~FitSpotsGraphics ()

10.9.1 Detailed Description

Graphics class for FittingSpots.

This abstraction prevents FitSpots from depending on and graphics library. The functions are tied very heavily to the internals of FitSpots.

Definition at line 21 of file multispot5.h.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 FitSpotsGraphics::~FitSpotsGraphics ( ) [virtual]

Desctructor.

Definition at line 71 of file multispot5.cc.

```
{}
```

10.9.3 Member Function Documentation

10.9.3.1 virtual void FitSpotsGraphics::init (CVD::ImageRef *size*) [pure virtual]

Initialize graphics.

Parameters

size	Size of window for display

Implemented in NullGraphics.

10.9.3.2 virtual void FitSpotsGraphics::draw_krap (const std::vector< TooN::Vector< 4 > > & *spots*, const CVD::Image< CVD::byte > & *im*, const BBox & *box*, int *N*, TooN::Vector< 4 > *s* = TooN::Ones *1e99) [pure virtual]

Draw a bunch of stuff.

Parameters

spots	List of spots to draw
im	Background image
box	Bounding box of region
N	Spot to highlight
s	Extra spot to draw as a cross

Implemented in NullGraphics.

10.9.3.3 virtual void FitSpotsGraphics::swap ( ) [pure virtual]

Swap buffers if double buffered.

Implemented in NullGraphics, and GraphicsGL.

10.9.3.4 virtual void FitSpotsGraphics::draw_pixels (const std::vector< CVD::ImageRef > & *pix*, float *r*, float *g*, float *b*, float *a* = 1) [pure virtual]

Draw the pixel mask in an (r,g,b,a) tuple colour.

Parameters

pix	mask
r	red
g	green
b	blue
a	alpha

Implemented in NullGraphics.

10.9.3.5 virtual void FitSpotsGraphics::draw_bbox (const BBox & *bbox*) [pure virtual]

Draw a bounding box.

Parameters

bbox	Box corners

Implemented in NullGraphics, and GraphicsGL.

10.9.3.6 virtual void FitSpotsGraphics::glDrawCross (const TooN::Vector< 2 > & *p*, int *size* = 3) [pure virtual]

Draw a cross.

Parameters

p	Position of cross
size	Size to draw cross

Implemented in NullGraphics.

The documentation for this class was generated from the following files:

- multispot5.h
- multispot5.cc

10.10 FloatSliderWithBox Class Reference

This class makes a floating point slider bar with an edit box next to it for more precision.

Classes

- class SliderChanged
- class TextChanged

Public Member Functions

- FloatSliderWithBox (String text_, double min_, double max_, double value_, int cols, boolean rec_)
- FloatSliderWithBox setUnits (String s)
- FloatSliderWithBox setFormat (String s)
- void addChangeListener (ChangeListener changeListener)
- double get_value_from_slider ()
- double get_value_from_text ()

Package Functions

- void setValue (double v)
- double getValue ()

Package Attributes

- double max

Private Attributes

- JSlider slider
- JTextField number
- JLabel label
- GridBagConstraints completePanelConstraints_
- int steps = 1000000
- double min
- String text

- String units
- String format = "%8.3f"
- double value
- boolean reciprocal

10.10.1 Detailed Description

This class makes a floating point slider bar with an edit box next to it for more precision.

Also has a reciprocal option for inverse, reciprocal scaling./

Definition at line 685 of file three_B.java.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 FloatSliderWithBox::FloatSliderWithBox (String *text_*, double *min_*, double *max_*, double *value_*, int *cols*, boolean *rec_*) [inline]

Definition at line 701 of file three_B.java.

References completePanelConstraints_, label, max, min, number, reciprocal, setValue(), slider, steps, text, and value.

```
{

    super( new GridBagLayout() );

    reciprocal = rec_;
    min=min_;
    max=max_;
    text=text_;
    value = value_;

    if(reciprocal)
    {
        min=1/max_;
        max=1/min_;
    }

    slider = new JSlider(0, steps);

    label = new JLabel();

    number = new JTextField(cols);

    //Assemble into a panel
    completePanelConstraints_ = new GridBagConstraints();
    completePanelConstraints_.fill = GridBagConstraints.HORIZONTAL;

    completePanelConstraints_.gridx = 0;
    completePanelConstraints_.gridy = 0;
    completePanelConstraints_.weightx=1;
    this.add( slider, completePanelConstraints_ );
```

```
        completePanelConstraints_.gridx = 2;
        completePanelConstraints_.gridy = 0;
        completePanelConstraints_.weightx=0;
        this.add(label, completePanelConstraints_ );

        completePanelConstraints_.gridx = 1;
        completePanelConstraints_.gridy = 0;
        completePanelConstraints_.weightx=0;
        //Add some space to the left to move away from slider slightly
        //And some more to the bottom to help with the way they are displayed
        completePanelConstraints_.insets = new Insets(0,5,10,0);
        this.add( number, completePanelConstraints_ );

        this.setBorder(BorderFactory.createTitledBorder(text));


        slider.addChangeListener(new SliderChanged(this));
        number.addActionListener(new TextChanged(this));
        setValue(value);

    }
```

10.10.3 Member Function Documentation

10.10.3.1 FloatSliderWithBox FloatSliderWithBox::setUnits (String *s*) [inline]

Definition at line 755 of file three_B.java.

References setValue(), units, and value.

Referenced by EControlPanel::EControlPanel().

```
    {
        units = s;
        setValue(value);
        return this;
    }
```

10.10.3.2 FloatSliderWithBox FloatSliderWithBox::setFormat (String *s*) [inline]

Definition at line 762 of file three_B.java.

References format, setValue(), and value.

Referenced by EControlPanel::EControlPanel().

```
    {
        format = s;
        setValue(value);
        return this;
    }
```

10.10.3.3 void FloatSliderWithBox::addChangeListener (ChangeListener *changeListener*) [inline]

Definition at line 769 of file three_B.java.

References slider.

Referenced by EControlPanel::EControlPanel().

```
                                                                    {
        slider.addChangeListener( changeListener );
        return;
    }
```

10.10.3.4 void FloatSliderWithBox::setValue (double *v*) [inline, package]

Definition at line 774 of file three_B.java.

References format, label, max, min, number, reciprocal, slider, steps, units, and value.

Referenced by FloatSliderWithBox::TextChanged::actionPerformed(), FloatSliderWithBox(), setFormat(), setUnits(), and FloatSliderWithBox::SliderChanged::stateChanged().

```
    {
        value = v;
        if(reciprocal)
            slider.setValue((int)Math.round(steps * (1/value-min)/(max-min)));
        else
            slider.setValue((int)Math.round(steps * (value-min)/(max-min)));

        number.setText(String.format(format, value));
        label.setText(units);
    }
```

10.10.3.5 double FloatSliderWithBox::getValue ( ) [inline, package]

Definition at line 786 of file three_B.java.

References value.

Referenced by EControlPanel::export_reconstruction_as_ij(), and EControlPanel::update_canvas().

```
    {
        return value;
    }
```

10.10.3.6 double FloatSliderWithBox::get_value_from_slider ( ) [inline]

Definition at line 791 of file three_B.java.

References max, min, reciprocal, slider, and steps.

Referenced by FloatSliderWithBox::SliderChanged::stateChanged().

```
{
    if(reciprocal)
        return 1/((slider.getValue() * 1.0 / steps) * (max - min) + min);
    else
        return (slider.getValue() * 1.0 / steps) * (max - min) + min;
}
```

10.10.3.7 double FloatSliderWithBox::get_value_from_text ( ) [inline]

Definition at line 798 of file three_B.java.

References number.

Referenced by FloatSliderWithBox::TextChanged::actionPerformed().

```
{
    return Double.parseDouble(number.getText());
}
```

10.10.4 Member Data Documentation

10.10.4.1 JSlider FloatSliderWithBox::slider [private]

Definition at line 687 of file three_B.java.

Referenced by addChangeListener(), FloatSliderWithBox(), get_value_from_slider(), and setValue().

10.10.4.2 JTextField FloatSliderWithBox::number [private]

Definition at line 688 of file three_B.java.

Referenced by FloatSliderWithBox(), get_value_from_text(), and setValue().

10.10.4.3 JLabel FloatSliderWithBox::label [private]

Definition at line 689 of file three_B.java.

Referenced by FloatSliderWithBox(), and setValue().

10.10.4.4 GridBagConstraints FloatSliderWithBox::completePanelConstraints_ [private]

Definition at line 690 of file three_B.java.

Referenced by FloatSliderWithBox().

10.10.4.5 int FloatSliderWithBox::steps = 1000000 [private]

Definition at line 692 of file three_B.java.

Referenced by FloatSliderWithBox(), get_value_from_slider(), and setValue().

10.10.4.6 double FloatSliderWithBox::min [private]

Definition at line 693 of file three_B.java.

Referenced by FloatSliderWithBox(), get_value_from_slider(), and setValue().

10.10.4.7 double FloatSliderWithBox::max [package]

Definition at line 693 of file three_B.java.

Referenced by FloatSliderWithBox(), get_value_from_slider(), and setValue().

10.10.4.8 String FloatSliderWithBox::text [private]

Definition at line 694 of file three_B.java.

Referenced by FloatSliderWithBox().

10.10.4.9 String FloatSliderWithBox::units [private]

Definition at line 695 of file three_B.java.

Referenced by setUnits(), and setValue().

10.10.4.10 String FloatSliderWithBox::format = "%8.3f" [private]

Definition at line 696 of file three_B.java.

Referenced by setFormat(), and setValue().

10.10.4.11 double FloatSliderWithBox::value [private]

Definition at line 698 of file three_B.java.

Referenced by FloatSliderWithBox(), getValue(), setFormat(), setUnits(), and setValue().

10.10.4.12 boolean FloatSliderWithBox::reciprocal [private]

Definition at line 699 of file three_B.java.

Referenced by FloatSliderWithBox(), get_value_from_slider(), and setValue().

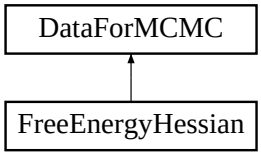
The documentation for this class was generated from the following file:

- three_B.java

10.11 FreeEnergyHessian Class Reference

Class for computing the Hessian of the negative free energy.

Inheritance diagram for FreeEnergyHessian:



Public Member Functions

- FreeEnergyHessian (const DataForMCMC &d)
- Matrix< 4 > hessian (const vector< Vector< 4 > > &spots, int spot) const
- MT19937 & get_rng () const

Protected Attributes

- const vector< ImageRef > & pixels
- const vector< vector< double > > & pixel_intensities
- const double mu_brightness
- const double sigma_brightness
- const double mu_blur
- const double sigma_blur
- const double variance
- const int samples
- const int sample_iterations

- const Matrix< 3 > A
- const Vector< 3 > pi
- MT19937 & rng

10.11.1 Detailed Description

Class for computing the Hessian of the negative free energy.

Definition at line 608 of file multispot5.cc.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 FreeEnergyHessian::FreeEnergyHessian (const DataForMCMC & *d*) [inline]

Constructor.

Parameters

d	All data required

Definition at line 614 of file multispot5.cc.

```
:DataForMCMC(d)
{
}
```

10.11.3 Member Function Documentation

10.11.3.1 Matrix<4> FreeEnergyHessian::hessian (const vector< Vector< 4 > > & *spots*, int *spot*) const [inline]

Compute the Hessian.

Parameters

spots	All spot positions
spot	spot to compute hessian for

Definition at line 622 of file multispot5.cc.

References SampledMultispot::compute_spot_intensity(), SampledMultispot::compute_spot_intensity_hessian(), diff_log_log_normal(), hess_log_log_normal(), SampledMultispot::GibbsSampler::next(), SampledMultispot::GibbsSampler::sample(), and SampledMultispot::GibbsSampler::samp intensities().

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes().

```
{
    cout << "----Computing pure MCMC hessian\n";
    const unsigned int nspots  = spots.size();
    const unsigned int nframes = pixel_intensities.size();
    const unsigned int npixels = pixels.size();
    cout << spot << " " << nspots << " " << nframes << " " << npixels << endl
  ;

    vector<vector<double> > spot_intensity; //[spot][pixel]
    for(unsigned int i=0; i < nspots; i++)
        spot_intensity.push_back(compute_spot_intensity(pixels, spots[i]));

    vector<tuple<double, Vector<4>, Matrix<4> > > spot_hess_etc =
  compute_spot_intensity_hessian(pixels, spots[spot]);

    GibbsSampler sampler(pixel_intensities, spot_intensity, spots, A, pi,
  variance, sample_iterations);

    //Compute derivative of log probability, summed (ie integrated)
    Matrix<4> sum_hess1 = Zeros(spots.size());
    Matrix<4> sum_hess2 = Zeros(spots.size());
    Vector<4> sum_deriv = Zeros(spots.size());

    for(int sample=0; sample < samples; sample++)
    {
        sampler.next(rng);

        //Compute d log P(data | x, model) / d model, for a given sample
        //And the hessian
        Matrix<4> hess = Zeros(spots.size());
        Vector<4> deriv = Zeros(spots.size());
        for(unsigned int frame=0; frame < nframes; frame++)
        {
            for(unsigned int pixel=0; pixel < npixels; pixel++)
            {
                double e = pixel_intensities[frame][pixel] - sampler.sample_i
  ntensities()[frame][pixel];
                //Build up the derivative
                if(sampler.sample()[spot][frame] == 0)
                {
                    hess += e * get<2>(spot_hess_etc[pixel]) - get<1>(spot_he
  ss_etc[pixel]).as_col() * get<1>(spot_hess_etc[pixel]).as_row();
                    deriv += e * get<1>(spot_hess_etc[pixel]);

                }
            }
        }

        hess[0][0] += hess_log_log_normal(spots[spot][0], mu_brightness,
  sigma_brightness);
        hess[1][1] += hess_log_log_normal(spots[spot][1], mu_blur,
  sigma_blur);
        sum_hess1 += hess;

        deriv[0] += diff_log_log_normal(spots[spot][0], mu_brightness,
  sigma_brightness);
        deriv[1] += diff_log_log_normal(spots[spot][1], mu_blur, sigma_blur);


        sum_hess2 += deriv.as_col() * deriv.as_row();
```

```
        sum_deriv = sum_deriv + deriv;
    }

    sum_hess1 /= (samples * variance);
    sum_hess2 /= (samples * variance);
    sum_deriv /= (samples * variance);


    cout << sum_hess1 << endl;
    cout << sum_hess2 << endl;
    cout << sum_deriv.as_col() * sum_deriv.as_row() <<  endl;

    cout << "......." << sum_deriv << endl;
    //Put in the prior
    //The derivative prior parts cancel out.
    //Rather sensibly this means that the second derivatives can be
    //computed without reference to the prior, and then the prior can
    //be added in later, i.e.: P(data, parameters) = P(data|parameter) P(para
  meters)

    //The second derivatives have been constructed to be diagonal
    DiagonalMatrix<4> hess_prior = Zeros(spots.size());

    cout << "sum of parts = \n" << sum_hess1 + sum_hess2 - sum_deriv.as_col()
   * sum_deriv.as_row() << endl;
    //TooN cannot currently add DiagonalMatrix to Matrix!!
    //sum_hess.diagonal_slice() += hess_prior.diagonal_slice();

    cout << "++++Done Computing pure MCMC hessian\n";
    return sum_hess1 + sum_hess2 - sum_deriv.as_col() * sum_deriv.as_row();
}
```

10.11.3.2 MT19937& DataForMCMC::get_rng ( ) const [inline, inherited]

Definition at line 189 of file multispot5.cc.

Referenced by NegativeFreeEnergy::compute_with_mask(), and NegativeFreeEnergy::operator()().

```
{
    return rng;
}
```

10.11.4 Member Data Documentation

10.11.4.1 const vector<ImageRef>& DataForMCMC::pixels [protected, inherited]

Definition at line 179 of file multispot5.cc.

10.11.4.2 const vector<vector<double> >& DataForMCMC::pixel_intensities [protected, inherited]

Definition at line 180 of file multispot5.cc.

10.11.4.3 const double DataForMCMC::mu_brightness [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.11.4.4 const double DataForMCMC::sigma_brightness [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.11.4.5 const double DataForMCMC::mu_blur [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.11.4.6 const double DataForMCMC::sigma_blur [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.11.4.7 const double DataForMCMC::variance [protected, inherited]

Definition at line 182 of file multispot5.cc.

10.11.4.8 const int DataForMCMC::samples [protected, inherited]

Definition at line 183 of file multispot5.cc.

10.11.4.9 const int DataForMCMC::sample_iterations [protected, inherited]

Definition at line 183 of file multispot5.cc.

10.11.4.10 const Matrix<3> DataForMCMC::A [protected, inherited]

Definition at line 184 of file multispot5.cc.

10.11.4.11 const Vector<3> DataForMCMC::pi [protected, inherited]

Definition at line 185 of file multispot5.cc.

10.11.4.12 MT19937& DataForMCMC::rng [protected, inherited]

Definition at line 186 of file multispot5.cc.

The documentation for this class was generated from the following file:

- multispot5.cc

10.12 SampledMultispot::GibbsSampler Class Reference

Draw samples from the spot states given the spots positions and some data.

`#include <sampled_multispot.h>`

Public Member Functions

- GibbsSampler (const vector< vector< double > > &pixel_intensities_, const vector< vector< double > > &spot_intensities_, const vector< Vector< 4 > > &spots_, const Matrix< 3 > A_, const Vector< 3 > pi_, double variance_, int sample_iterations_)
- void set_variance (double v)
- void reset ()
- template<class T >
  void next (T &rng)
- const vector< vector< State > > & sample () const
- const vector< vector< double > > & sample_intensities () const

Private Attributes

- const vector< vector< double > > & pixel_intensities
- const vector< vector< double > > & spot_intensities
- const vector< Vector< 4 > > spots
- const Matrix< 3 > A
- const Vector< 3 > pi
- const double base_variance
- double variance
- const int sample_iterations
- const int num_frames
- const int num_pixels
- const vector< int > O
- vector< vector< State > > current_sample
- vector< vector< double > > current_sample_intensities

10.12.1 Detailed Description

Draw samples from the spot states given the spots positions and some data.

Variable naming matches that in FitSpots.

Definition at line 188 of file sampled_multispot.h.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 SampledMultispot::GibbsSampler::GibbsSampler (const vector< vector< double > > & *pixel_intensities_*, const vector< vector< double > > & *spot_intensities_*, const vector< Vector< 4 > > & *spots_*, const Matrix< 3 > *A_*, const Vector< 3 > *pi_*, double *variance_*, int *sample_iterations_*) [inline]

Definition at line 207 of file sampled_multispot.h.

References assert_same_size(), and spot_intensities.

```
:pixel_intensities(pixel_intensities_), //pixel_intensities: [frame][pixels]
 spot_intensities(spot_intensities_),   //spot_intensities: [spot][pixel]
 spots(spots_),
 A(A_),
 pi(pi_),
 base_variance(variance_),
 variance(variance_),
 sample_iterations(sample_iterations_),
 num_frames(pixel_intensities.size()),
 num_pixels(pixel_intensities[0].size()),
    //Observations vector. As usual for this application, the observations ar
  e just
    //numbered integers which refer to data held elsewhere.
 O(sequence(num_frames)),
    //Start all spots OFF, so the intensity is 0. OFF is 1 or 2, not 0!!!
    //sample_list: [sample][spot][frame]: list of samples drawn using Gibbs s
  ampling
 current_sample(spots.size(), vector<State>(num_frames, 2)), //current sample
   [spot][frame]
    //pixel intensities assosciated with the current sample [frame][pixel]
 current_sample_intensities(num_frames, vector<double>(num_pixels))
{
    //Check a bunch of stuff
    assert_same_size(pixel_intensities);
    assert_same_size(spot_intensities);

}
```

10.12.3 Member Function Documentation

10.12.3.1 void SampledMultispot::GibbsSampler::set_variance (double *v*) [inline]

Update the noide variance.

Used for adding thermal noise.

Parameters

v	noise variance.

Definition at line 241 of file sampled_multispot.h.

Referenced by NegativeFreeEnergy::operator()().

```
{
    variance = v;
}
```

10.12.3.2 void SampledMultispot::GibbsSampler::reset ( ) [inline]

Reset the gibbs sampler oro the initial state (all spots off)

Definition at line 248 of file sampled_multispot.h.

```
{
    vector<State> off(num_frames, 2);
    fill(current_sample.begin(), current_sample.end(), off);

    vector<double> black(num_pixels);
    fill(current_sample_intensities.begin(), current_sample_intensities.end()
  , black);
    variance = base_variance;
}
```

10.12.3.3 template<class T > void SampledMultispot::GibbsSampler::next (T & *rng*) [inline]

Get the next sample.

Parameters

rng	Random number generator

Definition at line 260 of file sampled_multispot.h.

References SampledMultispot::add_spot(), forward_algorithm_delta(), SampledMultispot::remove_-spot(), and spot_intensities.

Referenced by FreeEnergyHessian::hessian(), NegativeFreeEnergy::operator()(), and FitSpots::try_modifying_model().

```
{
    for(int j=0; j < sample_iterations; j++)
        for(int k=0; k < (int) spots.size(); k++)
        {
```

```
            //Subtract off the spot we're interested in.
            remove_spot(current_sample_intensities, spot_intensities[k],
  current_sample[k]);

            //Now current_sample_intensities is the image value for every spo
  t in every frame,
            //except the current spot, which is always set to off. This allow
  s us to add it in
            //easily.
            SpotWithBackground B(current_sample_intensities,
  spot_intensities[k], pixel_intensities, variance);
            vector<array<double, 3> > delta = forward_algorithm_delta(A, pi,
  B, O);
            current_sample[k] = backward_sampling<3,State, T>(A, delta, rng);


            //Put the newly sampled spot in
            add_spot(current_sample_intensities, spot_intensities[k],
  current_sample[k]);
          }
  }
```

10.12.3.4 const vector<vector<State> >& SampledMultispot::GibbsSampler::sample ( ) const [inline]

Retrieve the current sample.

Definition at line 286 of file sampled_multispot.h.

Referenced by FreeEnergyHessian::hessian(), and FitSpots::try_modifying_model().

```
{
    return current_sample;
}
```

10.12.3.5 const vector<vector<double> >& SampledMultispot::GibbsSampler::sample_intensities ( ) const [inline]

Retrieve the intensities for the current sample.

Definition at line 291 of file sampled_multispot.h.

Referenced by FreeEnergyHessian::hessian(), NegativeFreeEnergy::operator()(), and FitSpots::try_modifying_model().

```
{
    return current_sample_intensities;
}
```

10.12.4 Member Data Documentation

10.12.4.1 const vector<vector<double> >& SampledMultispot::GibbsSampler::pixel_intensities [private]

Definition at line 190 of file sampled_multispot.h.

10.12.4.2 const vector<vector<double> >& SampledMultispot::GibbsSampler::spot_intensities [private]

Definition at line 191 of file sampled_multispot.h.

10.12.4.3 const vector<Vector<4> > SampledMultispot::GibbsSampler::spots [private]

Definition at line 192 of file sampled_multispot.h.

10.12.4.4 const Matrix<3> SampledMultispot::GibbsSampler::A [private]

Definition at line 193 of file sampled_multispot.h.

10.12.4.5 const Vector<3> SampledMultispot::GibbsSampler::pi [private]

Definition at line 194 of file sampled_multispot.h.

10.12.4.6 const double SampledMultispot::GibbsSampler::base_variance [private]

Definition at line 195 of file sampled_multispot.h.

10.12.4.7 double SampledMultispot::GibbsSampler::variance [private]

Definition at line 196 of file sampled_multispot.h.

10.12.4.8 const int SampledMultispot::GibbsSampler::sample_iterations [private]

Definition at line 198 of file sampled_multispot.h.

10.12.4.9 const int SampledMultispot::GibbsSampler::num_frames [private]

Definition at line 199 of file sampled_multispot.h.

10.12.4.10 const int SampledMultispot::GibbsSampler::num_pixels [private]

Definition at line 199 of file sampled_multispot.h.

10.12.4.11 const vector<int> SampledMultispot::GibbsSampler::O [private]

Definition at line 200 of file sampled_multispot.h.

10.12.4.12 vector<vector<State> > SampledMultispot::GibbsSampler::current_sample [private]

Definition at line 202 of file sampled_multispot.h.

10.12.4.13 vector<vector<double> > SampledMultispot::GibbsSampler::current_sample_intensities [private]

Definition at line 203 of file sampled_multispot.h.

The documentation for this class was generated from the following file:

- sampled_multispot.h

10.13 SampledMultispot::GibbsSampler2 Class Reference

Gibbs sampling class which masks spots to reduce computation.

`#include <sampled_multispot.h>`

Public Member Functions

- GibbsSampler2 (const vector< vector< double > > &pixel_intensities_, const vector< vector< double > > &spot_intensities_, const vector< Vector< 4 > > &spots_, const vector< vector< int > > &spot_pixels_, const Matrix< 3 > A_, const Vector< 3 > pi_, double variance_, int sample_iterations_)
- void set_variance (double v)
- void reset ()
- template<class T >
  void next (T &rng)

- const vector< vector< State > > & sample () const
- const vector< vector< double > > & sample_intensities () const

Private Attributes

- const vector< vector< double > > & pixel_intensities
- const vector< vector< double > > & spot_intensities
- const vector< Vector< 4 > > spots
- const std::vector< std::vector< int > > & spot_pixels
- const Matrix< 3 > A
- const Vector< 3 > pi
- const double base_variance
- double variance
- const int sample_iterations
- const int num_frames
- const int num_pixels
- const vector< int > O
- vector< vector< State > > current_sample
- vector< vector< double > > current_sample_intensities
- vector< double > cutout_spot_intensities
- vector< vector< double > > cutout_pixel_intensities
- vector< vector< double > > cutout_current_sample_intensities

10.13.1 Detailed Description

Gibbs sampling class which masks spots to reduce computation.

This draws samples from, the spot states given the spots positions and some data. It is very similar to GibbsSampler, except that it only computes probabilities in a mask around each spot to save on computation. Variable naming matches that in FitSpots.

Definition at line 304 of file sampled_multispot.h.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 SampledMultispot::GibbsSampler2::GibbsSampler2 (const vector< vector< double > > & *pixel_intensities_*, const vector< vector< double > > & *spot_intensities_*, const vector< Vector< 4 > > & *spots_*, const vector< vector< int > > & *spot_pixels_*, const Matrix< 3 > *A_*, const Vector< 3 > *pi_*, double *variance_*, int *sample_iterations_*) [inline]

Definition at line 328 of file sampled_multispot.h.

References assert_same_size(), and spot_intensities.

```
:pixel_intensities(pixel_intensities_), //pixel_intensities: [frame][pixels]
 spot_intensities(spot_intensities_),   //spot_intensities: [spot][pixel]
 spots(spots_),
 spot_pixels(spot_pixels_),
 A(A_),
 pi(pi_),
 base_variance(variance_),
 variance(variance_),
 sample_iterations(sample_iterations_),
 num_frames(pixel_intensities.size()),
 num_pixels(pixel_intensities[0].size()),
    //Observations vector. As usual for this application, the observations ar
  e just
    //numbered integers which refer to data held elsewhere.
 O(sequence(num_frames)),
    //Start all spots OFF, so the intensity is 0. OFF is 1 or 2, not 0!!!
    //sample_list: [sample][spot][frame]: list of samples drawn using Gibbs s
  ampling
 current_sample(spots.size(), vector<State>(num_frames, 2)), //current sample
   [spot][frame]
    //pixel intensities assosciated with the current sample [frame][pixel]
 current_sample_intensities(num_frames, vector<double>(num_pixels)),
 cutout_pixel_intensities(num_frames),
 cutout_current_sample_intensities(num_frames)
{
    //Check a bunch of stuff
    assert_same_size(pixel_intensities);
    assert_same_size(spot_intensities);
}
```

10.13.3 Member Function Documentation

10.13.3.1 void SampledMultispot::GibbsSampler2::set_variance (double *v*) [inline]

Update the noide variance.

Used for adding thermal noise.

Parameters

v	noise variance.

Definition at line 365 of file sampled_multispot.h.

Referenced by NegativeFreeEnergy::compute_with_mask().

```
{
    variance = v;
}
```

10.13.3.2 void SampledMultispot::GibbsSampler2::reset ( ) [inline]

Reset the gibbs sampler oro the initial state (all spots off)

Definition at line 372 of file sampled_multispot.h.

```
    {
        vector<State> off(num_frames, 2);
        fill(current_sample.begin(), current_sample.end(), off);

        vector<double> black(num_pixels);
        fill(current_sample_intensities.begin(), current_sample_intensities.end()
      , black);
        variance = base_variance;
    }
```

10.13.3.3 template<class T > void SampledMultispot::GibbsSampler2::next (T & *rng*) [inline]

Get the next sample.

Parameters

rng	Random number generator

Definition at line 384 of file sampled_multispot.h.

References SampledMultispot::add_spot(), SampledMultispot::remove_spot(), and spot_-intensities.

Referenced by NegativeFreeEnergy::compute_with_mask(), FitSpots::optimize_each_-spot_in_turn_for_several_passes(), and FitSpots::try_modifying_model().

```
    {

//double remove=0;
//double cut=0;
//double swb=0;
//double ff_masked=0;
//double bs=0;
//double add=0;
//cvd_timer t;
    std::vector<array<double, 3> > delta3;
        for(int j=0; j < sample_iterations; j++)
            for(int k=0; k < (int) spots.size(); k++)
            {
//t.reset();
                //Subtract off the spot we're interested in.
                remove_spot(current_sample_intensities, spot_intensities[k],
      current_sample[k], spot_pixels[k]);
//remove+=t.reset();
/*
                //Cut out
                //spot
                cutout_spot_intensities.resize(spot_pixels[k].size());
                for(unsigned int i=0; i < spot_pixels[k].size(); i++)
                    cutout_spot_intensities[i] = spot_intensities[k][spot_pixels[
      k][i]];

                //others
```

```
                for(int f=0; f < num_frames; f++)
                {
                    cutout_current_sample_intensities[f].resize(spot_pixels[k].si
      ze());
                    cutout_pixel_intensities[f].resize(spot_pixels[k].size());
                    for(unsigned int i=0; i < spot_pixels[k].size();i++)
                    {
                        cutout_current_sample_intensities[f][i] = current_sample_
      intensities[f][spot_pixels[k][i]];
                        cutout_pixel_intensities[f][i] = pixel_intensities[f][spo
      t_pixels[k][i]];
                    }
                }*/
//cut += t.reset();
                //Now current_sample_intensities is the image value for every spo
      t in every frame,
                //except the current spot, which is always set to off. This allow
      s us to add it in
                //easily.

//              SpotWithBackground B(current_sample_intensities, spot_intensities
      [k], pixel_intensities, variance);
//              vector<array<double, 3> > delta = forward_algorithm_delta(A, pi,
      B, O);

//ff+=t.reset();

//              SpotWithBackground B2(cutout_current_sample_intensities, cutout_s
      pot_intensities, cutout_pixel_intensities, variance);
//              std::vector<array<double, 3> > delta2 = forward_algorithm_delta(A
      , pi, B2, O);
//ff_cut+=t.reset();

                SpotWithBackgroundMasked B3(current_sample_intensities,
      spot_intensities[k], pixel_intensities, variance, spot_pixels[k]);
//swb += t.reset();
                forward_algorithm_delta2<3>(A, pi, B3, O, delta3);
//f_masked+=t.reset();
                /*for(unsigned int i=0; i < delta.size(); i++)
                {
                    cout.precision(20);
                    cout.setf(cout.scientific);
                    std::cout << delta[i][0] << " " << delta[i][1] << " " <<delta
      [i][2] << std::endl;
                    std::cout << delta2[i][0] << " " << delta2[i][1] << " " <<del
      ta2[i][2] << std::endl;
                    cout << endl;
                }
                std::exit(1);

*/

                current_sample[k] = backward_sampling<3,State, T>(A, delta3, rng)
      ;
//bs += t.reset();
                //Put the newly sampled spot in
                add_spot(current_sample_intensities, spot_intensities[k], current
      _sample[k], spot_pixels[k]);
//add += t.reset();
            }
//      cout << "remove=" <<remove << " cut=" << cut << " swb=" << swb<< " ff_mas
      k=" << ff_masked << " bs=" <<bs << " add="<<add << endl;
```

```
}
```

10.13.3.4 const vector<vector<State> >& SampledMultispot::GibbsSampler2::sample ( ) const [inline]

Retrieve the current sample.

Definition at line 465 of file sampled_multispot.h.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_modifying_model().

```
{
    return current_sample;
}
```

10.13.3.5 const vector<vector<double> >& SampledMultispot::GibbsSampler2::sample_intensities ( ) const [inline]

Retrieve the intensities for the current sample.

Definition at line 470 of file sampled_multispot.h.

Referenced by NegativeFreeEnergy::compute_with_mask(), FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_modifying_model().

```
{
    return current_sample_intensities;
}
```

10.13.4 Member Data Documentation

10.13.4.1 const vector<vector<double> >& SampledMultispot::GibbsSampler2::pixel_intensities [private]

Definition at line 306 of file sampled_multispot.h.

10.13.4.2 const vector<vector<double> >& SampledMultispot::GibbsSampler2::spot_intensities [private]

Definition at line 307 of file sampled_multispot.h.

10.13.4.3 const vector<Vector<4> > SampledMultispot::GibbsSampler2::spots [private]

Definition at line 308 of file sampled_multispot.h.

10.13.4.4 const std::vector<std::vector<int> >& SampledMultispot::GibbsSampler2::spot_pixels [private]

Definition at line 309 of file sampled_multispot.h.

10.13.4.5 const Matrix<3> SampledMultispot::GibbsSampler2::A [private]

Definition at line 310 of file sampled_multispot.h.

10.13.4.6 const Vector<3> SampledMultispot::GibbsSampler2::pi [private]

Definition at line 311 of file sampled_multispot.h.

10.13.4.7 const double SampledMultispot::GibbsSampler2::base_variance [private]

Definition at line 312 of file sampled_multispot.h.

10.13.4.8 double SampledMultispot::GibbsSampler2::variance [private]

Definition at line 313 of file sampled_multispot.h.

10.13.4.9 const int SampledMultispot::GibbsSampler2::sample_iterations [private]

Definition at line 315 of file sampled_multispot.h.

10.13.4.10 const int SampledMultispot::GibbsSampler2::num_frames [private]

Definition at line 316 of file sampled_multispot.h.

10.13.4.11 const int SampledMultispot::GibbsSampler2::num_pixels [private]

Definition at line 316 of file sampled_multispot.h.

10.13.4.12 const vector<int> SampledMultispot::GibbsSampler2::O [private]

Definition at line 317 of file sampled_multispot.h.

10.13.4.13 vector<vector<State> > SampledMultispot::GibbsSampler2::current_sample [private]

Definition at line 319 of file sampled_multispot.h.

10.13.4.14 vector<vector<double> > SampledMultispot::GibbsSampler2::current_sample_intensities [private]

Definition at line 320 of file sampled_multispot.h.

10.13.4.15 vector<double> SampledMultispot::GibbsSampler2::cutout_spot_intensities [private]

Definition at line 322 of file sampled_multispot.h.

10.13.4.16 vector<vector<double> > SampledMultispot::GibbsSampler2::cutout_pixel_intensities [private]

Definition at line 323 of file sampled_multispot.h.

10.13.4.17 vector<vector<double> > SampledMultispot::GibbsSampler2::cutout_current_sample_intensities [private]

Definition at line 324 of file sampled_multispot.h.

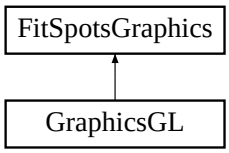
The documentation for this class was generated from the following file:

- sampled_multispot.h

10.14 GraphicsGL Class Reference

Graphics class which draws information to the screen using OpenGL.

Inheritance diagram for GraphicsGL:



Public Member Functions

- GraphicsGL ()
- virtual void draw_pixels (const vector< ImageRef > &pix, float r, float g, float b, float a)
- virtual void draw_bbox (const BBox &bbox)
- virtual void draw_krap (const vector< Vector< 4 > > &spots, const Image< byte > &im, const BBox &box, int N, Vector< 4 > s)
- virtual void glDrawCross (const Vector< 2 > &p, int size)
- virtual void swap ()
- virtual void init (ImageRef log_ratios_size)
- virtual ~GraphicsGL ()
- virtual void init (CVD::ImageRef size)=0
- virtual void draw_krap (const std::vector< TooN::Vector< 4 > > &spots, const CVD::Image< CVD::byte > &im, const BBox &box, int N, TooN::Vector< 4 > s=TooN::Ones *1e99)=0
- virtual void draw_pixels (const std::vector< CVD::ImageRef > &pix, float r, float g, float b, float a=1)=0
- virtual void glDrawCross (const TooN::Vector< 2 > &p, int size=3)=0

Private Member Functions

- GraphicsGL (const GraphicsGL &)
- void glDrawCircle (const Vector< 2 > &p, float r)
- void set_GL_zoom_size (ImageRef size, double scale)

Private Attributes

- std::auto_ptr< GLWindow > win
- int debug_window_zoom

10.14.1 Detailed Description

Graphics class which draws information to the screen using OpenGL.

Definition at line 119 of file multispot5_gui.cc.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 GraphicsGL::GraphicsGL (const GraphicsGL &) [private]

10.14.2.2 GraphicsGL::GraphicsGL ( ) [inline]

Definition at line 172 of file multispot5_gui.cc.

```
{
    debug_window_zoom = GV3::get<int>("debug.zoom", 3, 1);
}
```

10.14.2.3 virtual GraphicsGL::~GraphicsGL ( ) [inline, virtual]

Definition at line 254 of file multispot5_gui.cc.

```
{
}
```

10.14.3 Member Function Documentation

10.14.3.1 void GraphicsGL::glDrawCircle (const Vector< 2 > & *p*, float *r*) [inline, private]

Generate circle linesegments as pair of vertices for GL_LINES.

Parameters

p	circle centre
r	circle radius

Definition at line 129 of file multispot5_gui.cc.

```
{
    float theta=0;
    for(;;)
    {
        glVertex(p + r * makeVector(cos(theta), sin(theta)));
        theta +=0.01;
        if(theta > M_PI*2)
            break;
        glVertex(p + r * makeVector(cos(theta), sin(theta)));
    }
    glVertex(p + r * makeVector(1, 0));
}
```

10.14.3.2 void GraphicsGL::set_GL_zoom_size (ImageRef *size*, double *scale*) [inline, private]

Set up a GL window so that glDrawPixels and glVertex line up and also so that glDrawPixels is zoomed.

Parameters

size	Window size
scale	Zoom level

Definition at line 147 of file multispot5_gui.cc.

```
{
    double right = size.x;
    double bottom = size.y;
    //double my_scale = scale;


    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();


    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();
    glOrtho(0-.5, right-.5, bottom-.5, -.5, -1 , 1);    // If the origin
is the top left

    glRasterPos2f(-.5, -.5);

    // video is now the same way as upside down to graphics!
    glPixelZoom(scale, -scale);
}
```

10.14.3.3 virtual void GraphicsGL::draw_pixels (const vector< ImageRef > & *pix*, float *r*, float *g*, float *b*, float *a*) [inline, virtual]

Definition at line 177 of file multispot5_gui.cc.

```
{
    glColor4f(r, g, b, a);
    glPointSize(1.5);
    glBegin(GL_POINTS);
    glVertex(pix);
    glEnd();
}
```

10.14.3.4 virtual void GraphicsGL::draw_bbox (const BBox & *bbox*) [inline, virtual]

Draw a bounding box.

Parameters

bbox	Box corners

Implements FitSpotsGraphics.

Definition at line 186 of file multispot5_gui.cc.

References draw_bbox().

```
        {
::draw_bbox(bbox);
        }
```

10.14.3.5 virtual void GraphicsGL::draw_krap (const vector< Vector< 4 > > & *spots*, const Image< byte > & *im*, const BBox & *box*, int *N*, Vector< 4 > *s*) [inline, virtual]

Definition at line 192 of file multispot5_gui.cc.

References draw_bbox().

```
        {

            glDrawPixels(im);
            glColor3f(1, 0, 0);
            draw_bbox(box);

            glLineWidth(0.3);
            glBegin(GL_LINES);
            for(unsigned int i=0; i < spots.size(); i++)
            {
                glColor4f(0, 1, 0, .3);

                if((int)i == N && s[0] != 1e99)
                    glDrawCircle(s.slice<2, 2>(), s[1]);
                else
                    glDrawCircle(spots[i].slice<2, 2>(), spots[i][1]);
            }
            glEnd();

            glLineWidth(1.0);
            glBegin(GL_LINES);
            for(unsigned int i=0; i < spots.size(); i++)
            {
                glColor3f(1, 0, 0);
                if((int) i == N)
                    glColor3f(1, 1, 0);

                if((int)i == N && s[0] != 1e99)
                    glDrawCross(s.slice<2, 2>(), 1);
```

```
            else
                glDrawCross(spots[i].slice<2, 2>(), 1);
        }
        glEnd();

        glFlush();
    }
```

10.14.3.6 virtual void GraphicsGL::glDrawCross (const Vector< 2 > & *p*, int *size*) [inline, virtual]

Definition at line 230 of file multispot5_gui.cc.

```
    {
        glVertex(p + makeVector(0,  size));
        glVertex(p + makeVector(0, -size));
        glVertex(p + makeVector( size, 0));
        glVertex(p + makeVector(-size, 0));
    }
```

10.14.3.7 virtual void GraphicsGL::swap ( ) [inline, virtual]

Swap buffers if double buffered.

Implements FitSpotsGraphics.

Definition at line 239 of file multispot5_gui.cc.

```
    {
        win->swap_buffers();
    }
```

10.14.3.8 virtual void GraphicsGL::init (ImageRef *log_ratios_size*) [inline, virtual]

Definition at line 245 of file multispot5_gui.cc.

```
    {
        win = auto_ptr<GLWindow>(new GLWindow(log_ratios_size*
debug_window_zoom));
        set_GL_zoom_size(log_ratios_size, debug_window_zoom);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glEnable(GL_LINE_SMOOTH);
    }
```

10.14.3.9 virtual void FitSpotsGraphics::init (CVD::ImageRef *size*) [pure virtual, inherited]

Initialize graphics.

Parameters

size	Size of window for display

Implemented in NullGraphics.

10.14.3.10 virtual void FitSpotsGraphics::draw_krap (const std::vector< TooN::Vector< 4 > > & *spots*, const CVD::Image< CVD::byte > & *im*, const BBox & *box*, int *N*, TooN::Vector< 4 > *s* = TooN::Ones *1e99) [pure virtual, inherited]

Draw a bunch of stuff.

Parameters

spots	List of spots to draw
im	Background image
box	Bounding box of region
N	Spot to highlight
s	Extra spot to draw as a cross

Implemented in NullGraphics.

10.14.3.11 virtual void FitSpotsGraphics::draw_pixels (const std::vector< CVD::ImageRef > & *pix*, float *r*, float *g*, float *b*, float *a* = 1) [pure virtual, inherited]

Draw the pixel mask in an (r,g,b,a) tuple colour.

Parameters

pix	mask
r	red
g	green
b	blue
a	alpha

Implemented in NullGraphics.

10.14.3.12 virtual void FitSpotsGraphics::glDrawCross (const TooN::Vector< 2 > & *p*, int *size* = 3) [pure virtual, inherited]

Draw a cross.

Parameters

p	Position of cross
size	Size to draw cross

Implemented in NullGraphics.

10.14.4 Member Data Documentation

10.14.4.1 std::auto_ptr<GLWindow> GraphicsGL::win [private]

Definition at line 122 of file multispot5_gui.cc.

10.14.4.2 int GraphicsGL::debug_window_zoom [private]

Definition at line 124 of file multispot5_gui.cc.

The documentation for this class was generated from the following file:

- multispot5_gui.cc

10.15 IndexLexicographicPosition< Cmp, First > Struct Template Reference

Class for sorting a list of indexes to an array of spots lexicographically according to the 2D positions of the spots.

Public Member Functions

- IndexLexicographicPosition (const vector< Vector< 4 > > &s)
- bool operator() (int a, int b)

Public Attributes

- const vector< Vector< 4 > > & spots

Static Public Attributes

- static const int Second = First==2?3:2

10.15.1 Detailed Description

template<class Cmp, int First>struct IndexLexicographicPosition< Cmp, First >

Class for sorting a list of indexes to an array of spots lexicographically according to the 2D positions of the spots.

Parameters

Cmp	comparator function to specify less or greater
First	most significant position index

Definition at line 453 of file multispot5.cc.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 template<class Cmp , int First> IndexLexicographicPosition< Cmp, First >::IndexLexicographicPosition (const vector< Vector< 4 > > & *s*) [inline]

Parameters

s	Vector to sort indices of

Definition at line 457 of file multispot5.cc.

```
:spots(s)
{}
```

10.15.3 Member Function Documentation

10.15.3.1 template<class Cmp , int First> bool IndexLexicographicPosition< Cmp, First >::operator() (int *a*, int *b*) [inline]

Compare two indexes into the array of spots.

Definition at line 465 of file multispot5.cc.

```
{
    Cmp cmp;

    if(cmp(spots[a][First], spots[b][First]))
        return true;
    else if(spots[a][First] == spots[b][First])
        return cmp(spots[a][Second], spots[b][Second]);
    else
        return false;
}
```

10.15.4 Member Data Documentation

10.15.4.1 template<class Cmp , int First> const vector<Vector<4> >& IndexLexicographicPosition< Cmp, First >::spots

Keep around the array of spots for later comprison.

Definition at line 454 of file multispot5.cc.

10.15.4.2 template<class Cmp , int First> const int IndexLexicographicPosition< Cmp, First >::Second = First==2?3:2 [static]

Second most siginifcant position index for sorting.

Definition at line 462 of file multispot5.cc.

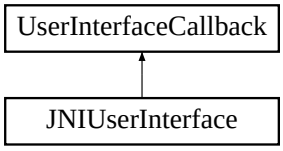
The documentation for this struct was generated from the following file:

- multispot5.cc

10.16 JNIUserInterface Class Reference

3B User interface for the Java plugin.

Inheritance diagram for JNIUserInterface:



Public Member Functions

- JNIUserInterface (JNIEnv *env_, jobject jthis)
- virtual void per_spot (int iteration, int pass, int spot_num, int total_spots)
- virtual void per_modification (int iteration, int spot_num, int total_spots)
- virtual void per_pass (int, int, const std::vector< TooN::Vector< 4 > > &spots)
- virtual void perhaps_stop ()
- void send_message (const string &s)
- void fatal (const string &s)

Private Attributes

- JNIEnv * env
- jobject ThreeBRunner_this
- jmethodID send_message_string
- jmethodID die
- jmethodID should_stop
- jmethodID send_new_points
- int passes

10.16.1 Detailed Description

3B User interface for the Java plugin.

This particular UI ferries various messages between Java and C++ via callbacks and polling.

Definition at line 33 of file multispot5_jni.cc.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 JNIUserInterface::JNIUserInterface (JNIEnv * *env_*, jobject *jthis*) [inline]

Definition at line 45 of file multispot5_jni.cc.

```
  :env(env_),ThreeBRunner_this(jthis)
  {
      jclass cls = env->GetObjectClass(jthis);

      send_message_string  = env->GetMethodID(cls, "send_message_string", "
(Ljava/lang/String;)V");
      die  = env->GetMethodID(cls, "die", "(Ljava/lang/String;)V");

      should_stop  = env->GetMethodID(cls, "should_stop", "()Z");

      send_new_points = env->GetMethodID(cls, "send_new_points", "([F)V");

      passes = GV3::get<int>("main.passes");
  }
```

10.16.3 Member Function Documentation

10.16.3.1 virtual void JNIUserInterface::per_spot (int *iteration*, int *pass*, int *spot_num*, int *total_spots*) [inline, virtual]

This function is called once per spot in each pass.

The idea is to provide a display along the lines of: Iteration #1 optimizing #2% complete

Parameters

iteration	Iteration number
pass	Pass number
spot_num	Spot currently being optimized
total_spots	Total number of spots to be optimized

Implements UserInterfaceCallback.

Definition at line 61 of file multispot5_jni.cc.

```
{
    send_message(sPrintf("Iteration %i, optimizing  %4i%%", iteration*
passes+pass, 100 *spot_num / total_spots));
}
```

10.16.3.2 virtual void JNIUserInterface::per_modification (int *iteration*, int *spot_num*, int *total_spots*) [inline, virtual]

This function is called once per spot in the modification phase.

The idea is to provide a display along the lines of: Iteration #1 modifying #2% complete

Parameters

iteration	Iteration number
spot_num	Spot currently being optimized
total_spots	Total number of spots to be optimized

Implements UserInterfaceCallback.

Definition at line 66 of file multispot5_jni.cc.

```
{
    send_message(sPrintf("Iteration %i, modifying  %4i%%", iteration*
passes+passes-1, 100 *spot_num / total_spots));
}
```

10.16.3.3 virtual void JNIUserInterface::per_pass (int *iteration*, int *pass*, const std::vector< TooN::Vector< 4 > > & *spots*) [inline, virtual]

This function is called once each time PASS data is outputted.

It will allow the GUI to build up a reconstruction.

Parameters

iteration	Iteration number
pass	Pass number
spots	Data to be reconstructed

Implements UserInterfaceCallback.

Definition at line 71 of file multispot5_jni.cc.

```
{
    //Copy data into the correct format
    vector<jfloat> pts_data;
    for(unsigned int i=0; i < spots.size(); i++)
    {
        pts_data.push_back(spots[i][2]);
        pts_data.push_back(spots[i][3]);
    }

    //Allocate a java array and copy data into it
    jfloatArray pts = env->NewFloatArray(pts_data.size());
    env->SetFloatArrayRegion(pts, 0, pts_data.size(), pts_data.data());

    //Make the call...
    jvalue pts_obj;
    pts_obj.l = pts;

    env->CallVoidMethod(ThreeBRunner_this, send_new_points, pts_obj);

    //Free the object
    env->DeleteLocalRef(pts);
}
```

10.16.3.4 virtual void JNIUserInterface::perhaps_stop ( ) [inline, virtual]

The user wishes to issue a stop instruction to the program (perhaps done via an asynchronus call to an instance of of UserInterfaceCallback).

This function is called as often as possible and will throw UserIssuedStop when the condition is met.

Implements UserInterfaceCallback.

Definition at line 95 of file multispot5_jni.cc.

```
{
    bool stop = env->CallBooleanMethod(ThreeBRunner_this, should_stop);
    if(stop)
        throw UserIssuedStop();
}
```

10.16.3.5 void JNIUserInterface::send_message (const string & *s*) [inline]

Definition at line 103 of file multispot5_jni.cc.

Referenced by Java_ThreeBRunner_call().

```
{
    jvalue message_string;
    message_string.l = env->NewStringUTF(s.c_str());
```

```
      env->CallVoidMethod(ThreeBRunner_this, send_message_string, message_s
tring);
      env->DeleteLocalRef(message_string.l);
  }
```

10.16.3.6 void JNIUserInterface::fatal (const string & *s*) [inline]

Definition at line 111 of file multispot5_jni.cc.

Referenced by Java_ThreeBRunner_call().

```
  {
      jvalue message_string;
      message_string.l = env->NewStringUTF(s.c_str());
      env->CallVoidMethod(ThreeBRunner_this, die, message_string);
      env->DeleteLocalRef(message_string.l);
  }
```

10.16.4 Member Data Documentation

10.16.4.1 JNIEnv* JNIUserInterface::env [private]

Definition at line 36 of file multispot5_jni.cc.

10.16.4.2 jobject JNIUserInterface::ThreeBRunner_this [private]

Definition at line 37 of file multispot5_jni.cc.

10.16.4.3 jmethodID JNIUserInterface::send_message_string [private]

Definition at line 38 of file multispot5_jni.cc.

10.16.4.4 jmethodID JNIUserInterface::die [private]

Definition at line 39 of file multispot5_jni.cc.

10.16.4.5 jmethodID JNIUserInterface::should_stop [private]

Definition at line 40 of file multispot5_jni.cc.

10.16.4.6 jmethodID JNIUserInterface::send_new_points [private]

Definition at line 41 of file multispot5_jni.cc.

10.16.4.7 int JNIUserInterface::passes [private]

Definition at line 42 of file multispot5_jni.cc.

The documentation for this class was generated from the following file:

- multispot5_jni.cc

10.17 Kahan Class Reference

Class implementing the Kahan summation algorithm to allow accurate summation of very large numbers of doubles.

Public Member Functions

- Kahan ()
- void add (double i)

Public Attributes

- double sum

Private Attributes

- double y
- double c
- double t

10.17.1 Detailed Description

Class implementing the Kahan summation algorithm to allow accurate summation of very large numbers of doubles.

Definition at line 226 of file multispot5.cc.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 Kahan::Kahan ( ) [inline]

Definition at line 234 of file multispot5.cc.

```
:c(0),sum(0)
{}
```

10.17.3 Member Function Documentation

10.17.3.1 void Kahan::add (double *i*) [inline]

Add a number to the running sum.

Parameters

i	Number to add

Definition at line 240 of file multispot5.cc.

Referenced by NegativeFreeEnergy::compute_with_mask(), and NegativeFreeEnergy::operator()().

```
{
    //y = input -c
    y = i;
    y-= c;

    //t = sum + y
    t = sum;
    t += y;

    //c = (t - sum) - y
    //c = ((sum + y) - sum) - y)
    c = t;
    c -= sum;
    c -= y;
    sum = t;
}
```

10.17.4 Member Data Documentation

10.17.4.1 double Kahan::y [private]

Input with the compensation removed. Temporary working space.

Definition at line 228 of file multispot5.cc.

10.17.4.2 double Kahan::c [private]

Running compenstation for low-order bits.

Definition at line 229 of file multispot5.cc.

10.17.4.3 double Kahan::t [private]

y + sum, which loses low order bits of y. Temporary working space.

Definition at line 230 of file multispot5.cc.

10.17.4.4 double Kahan::sum

running sum

Definition at line 232 of file multispot5.cc.

The documentation for this class was generated from the following file:

- multispot5.cc

10.18 LessSecond Struct Reference

Comparator functor for the first element of a std::pair.

Public Member Functions

- template<class A , class B >
  bool operator() (const pair< A, B > &a, const pair< A, B > &b) const

10.18.1 Detailed Description

Comparator functor for the first element of a std::pair.

Definition at line 707 of file multispot5.cc.

10.18.2 Member Function Documentation

10.18.2.1 template<class A , class B > bool LessSecond::operator() (const pair< A, B > & *a*, const pair< A, B > & *b*) const [inline]

Comparison function.

Parameters

a	
b	

Definition at line 712 of file multispot5.cc.

```
{
    return a.second < b.second;
}
```

The documentation for this struct was generated from the following file:

- multispot5.cc

10.19 LoadTestData Class Reference

Plugin class to load the standard test data from the JAR file.

Public Member Functions

- void run (String arg)

10.19.1 Detailed Description

Plugin class to load the standard test data from the JAR file.

Definition at line 27 of file LoadTestData.java.

10.19.2 Member Function Documentation

10.19.2.1 void LoadTestData::run (String *arg*) [inline]

Definition at line 29 of file LoadTestData.java.

```
{
    Opener o = new Opener();
    InputStream s = getClass().getClassLoader().getResourceAsStream("test_dat
  a.tif");
    ImagePlus im = o.openTiff(s, "test_data.tiff");


    try{
            s.close();
    }
    catch(IOException close_err){
        Toolkit.getDefaultToolkit().beep();
        ij.IJ.showStatus("Error reading image.");
        return;
    }
```

```
        im.show();
        im.updateAndDraw();


        System.out.println(getClass().getResource("").toString() + "\n");
        System.out.println(getClass().getResource("").getPath().toString() + "\n"
      );
        System.out.println(getClass().getResource("").getFile().toString() + "\n"
      );

    }
```

The documentation for this class was generated from the following file:

- LoadTestData.java

10.20 LogFileParseError Struct Reference

Null struct thrown if a parse error is encountered when trying to load a log file.

`#include <multispot5.h>`

Public Member Functions

- LogFileParseError (const std::string &s)

Public Attributes

- std::string what

10.20.1 Detailed Description

Null struct thrown if a parse error is encountered when trying to load a log file.

Definition at line 102 of file multispot5.h.

10.20.2 Constructor & Destructor Documentation

10.20.2.1 LogFileParseError::LogFileParseError (const std::string & *s*) [inline]

Parameters

s	error message to set

Definition at line 105 of file multispot5.h.

```
:what(s)
{}
```

10.20.3 Member Data Documentation

10.20.3.1 std::string LogFileParseError::what

variable holding the parse error error message

Definition at line 110 of file multispot5.h.

Referenced by mmain().

The documentation for this struct was generated from the following file:

- multispot5.h

10.21 MT19937 Struct Reference

Useful wrapper for MT19937 random number generator class.

`#include <mt19937.h>`

Classes

- struct ParseError

  Null struct thrown if attempting to load state from stream yields a parse error.

Public Member Functions

- MT19937 ()
- void simple_seed (int seed)
- void copy_state (const MT19937 &r)
- double operator() ()
- uint32_t rand_int ()
- double gaussian ()
- void write (std::ostream &o)
- void read (std::istream &is)

Public Attributes

- CRandomMersenne rng

Private Member Functions

- MT19937 (const MT19937 &)

10.21.1 Detailed Description

Useful wrapper for MT19937 random number generator class.

Definition at line 13 of file mt19937.h.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 MT19937::MT19937 ( ) [inline]

Definition at line 23 of file mt19937.h.

```
:rng(0)
{}
```

10.21.2.2 MT19937::MT19937 (const MT19937 &) [private]

Disallow copying, since one almost never wants to do this.

Copying has to be explicit via MT19937::copy_state().

10.21.3 Member Function Documentation

10.21.3.1 void MT19937::simple_seed (int *seed*) [inline]

Seed state with a simple RNG.

Parameters

seed	

Definition at line 29 of file mt19937.h.

References rng.

```
{
    rng.RandomInit(seed);
}
```

10.21.3.2 void MT19937::copy_state (const MT19937 & *r*) [inline]

Duplicate RNG state.

Parameters

r	RNG to duplicate

Definition at line 36 of file mt19937.h.

References rng.

```
{
    rng = r.rng;
}
```

10.21.3.3 double MT19937::operator() ( ) [inline]

Generate a double.

Definition at line 42 of file mt19937.h.

References rng.

```
{
    return rng.Random();
}
```

10.21.3.4 uint32_t MT19937::rand_int ( ) [inline]

Generate an int.

Definition at line 48 of file mt19937.h.

References rng.

```
{
    return rng.BRandom();
}
```

10.21.3.5 double MT19937::gaussian ( ) [inline]

Generate a Gaussian variate.

Definition at line 54 of file mt19937.h.

```
{
    double x1, x2, w;
    do {
```

```
            x1 = 2.0 * (*this)() - 1.0;
            x2 = 2.0 * (*this)() - 1.0;
            w = x1 * x1 + x2 * x2;
        } while ( w >= 1.0 );

        w = std::sqrt( (-2.0 * std::log( w ) ) / w );
        return x1 * w;
        //spare so we don't have to save that one extra bit of state y2 = x2
* w;
  }
```

10.21.3.6 void MT19937::write (std::ostream & *o*) [inline]

Serialise state.

Parameters

o	Stream to serialise to

Definition at line 70 of file mt19937.h.

References rng.

```
    {
        using namespace std;
        char f = o.fill();
        ios_base::fmtflags fl = o.flags();
        o << "MT19937 " << hex << setfill('0') << setw(3) << rng.get_index();

        for(int i=0; i < MERS_N; i++)
            o << " " << hex << setw(8) << rng.get_state()[i];

        o << setfill(f) << setiosflags(fl);
    }
```

10.21.3.7 void MT19937::read (std::istream & *is*) [inline]

De serialise state param is Stream to de-serialise from.

Definition at line 84 of file mt19937.h.

References rng.

```
    {
        using namespace std;

        string ls;
        getline(is, ls);
        if(ls.size() != 5627)
        {
            cerr << "MT19937: Expected string of length 5627. Got " << ls.siz
e() << endl;
            throw ParseError();
        }
```

```
        istringstream l(ls);

        string s;
        uint32_t i;

        l >> s;

        if(s != "MT19937")
        {
            cerr << "MT19937: Expected MT19937. Got " << s << endl;
            throw ParseError();
        }

        for(int n=0; n < MERS_N + 1; n++)
        {
            l >> hex >> i;
            if(l.bad())
            {
                cerr << "MT19937: Expected hex number. Got ";
                if(l.eof())
                    cerr << "EOF" << endl;
                else
                {
                    cerr << l.get() << endl;
                }

                throw ParseError();
            }

            if(n==0)
                rng.get_index() = i;
            else
                rng.get_state()[n-1]=i;

        }

    }
```

10.21.4 Member Data Documentation

10.21.4.1 CRandomMersenne MT19937::rng

Underlying RNG.

Definition at line 20 of file mt19937.h.

Referenced by copy_state(), operator()(), rand_int(), read(), simple_seed(), and write().

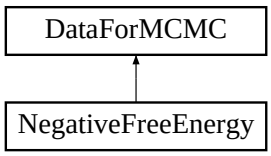
The documentation for this struct was generated from the following file:

- mt19937.h

10.22 NegativeFreeEnergy Class Reference

Class for computing the negitve free energy using thermodynamic integration.

Inheritance diagram for NegativeFreeEnergy:



Public Member Functions

- NegativeFreeEnergy (const DataForMCMC &d)
- double variance_from_sample (double sample, double samples, double base_sigma, double scale_pow) const
- double compute_with_mask (const Vector<> &spots, const vector< vector< int > > &spot_pixels) const
- double operator() (const Vector<> &spots) const
- MT19937 & get_rng () const

Protected Attributes

- const vector< ImageRef > & pixels
- const vector< vector< double > > & pixel_intensities
- const double mu_brightness
- const double sigma_brightness
- const double mu_blur
- const double sigma_blur
- const double variance
- const int samples
- const int sample_iterations
- const Matrix< 3 > A
- const Vector< 3 > pi
- MT19937 & rng

10.22.1 Detailed Description

Class for computing the negitve free energy using thermodynamic integration.

Definition at line 261 of file multispot5.cc.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 NegativeFreeEnergy::NegativeFreeEnergy (const DataForMCMC & *d*) [inline]

Parameters

d	Necessary data

Definition at line 266 of file multispot5.cc.

```
:DataForMCMC(d)
{
}
```

10.22.3 Member Function Documentation

10.22.3.1 double NegativeFreeEnergy::variance_from_sample (double *sample,* double *samples,* double *base_sigma,* double *scale_pow*) const [inline]

Give the noise variance given a sample number and growth parameters.

Parameters

sample	Sample number
samples	Total number of samples
base_sigma	Starting value of sigme
scale_pow	Exponent scaling

Definition at line 276 of file multispot5.cc.

References scale(), and sq().

```
{
    double scale = pow(1.25, sample * 1. / samples * scale_pow);
    double sigma = base_sigma * scale;
    double new_variance = sq(sigma);

    return new_variance;
}
```

10.22.3.2 double NegativeFreeEnergy::compute_with_mask (const Vector<> & *spots,* const vector< vector< int > > & *spot_pixels*) const [inline]

Estimate free energy using the Slow Growth Thermodynamic Integration method given in Probalistic Inference Using Markov Chain Monte Carlo Methods, Radford.

M. Neal, 1993 Except using a 5 point stencil instead of forward differenceing in Eq 6.30

Parameters

spots	list of spots
spot_pixels	Mask around each spot to use to save on computation

Definition at line 290 of file multispot5.cc.

References Kahan::add(), SampledMultispot::compute_spot_intensity(), DataForMCMC::get_-rng(), log_log_normal(), SampledMultispot::GibbsSampler2::next(), SampledMultispot::GibbsSampler2::sample_-intensities(), SampledMultispot::GibbsSampler2::set_variance(), spots_to_vector(), and sq().

Referenced by FitSpots::try_modifying_model().

```
{
    //Estimate free energy using the Slow Growth Thermodynamic Integration me
  thod given in
    //Probalistic Inference Using Markov Chain Monte Carlo Methods, Radford.
  M. Neal, 1993
    //Except using a 5 point stencil instead of forward differenceing in Eq 6
  .30
    double base_sigma = sqrt(variance); // should be 1
    double scale_pow = 100.0;

    const unsigned int nspots  = spots.size()/4;
    const unsigned int nframes = pixel_intensities.size();
    const unsigned int npixels = pixels.size();
    assert(spots.size() %4 == 0);
    assert(spot_pixels.size() == nspots);

    //Compute the intensities and derivatives for all spot
    vector<vector<double> > spot_intensity; //[spot][pixel]
    for(unsigned int i=0; i < nspots; i++)
        spot_intensity.push_back(compute_spot_intensity(pixels, spots.slice<D
  ynamic,4>(i*4,4)));

    GibbsSampler2 sampler(pixel_intensities, spot_intensity, spots_to_vector(
  spots), spot_pixels, A, pi, variance, sample_iterations);

    double sum = 0;
    Kahan ksum;
    for(int sample=0; sample < samples; sample++)
    {
        //Compute the positions of the surrounding steps
        double var1 = variance_from_sample(sample-2, samples, base_sigma, sca
  le_pow);
        double var2 = variance_from_sample(sample-1, samples, base_sigma, sca
  le_pow);
        double var3 = variance_from_sample(sample+1, samples, base_sigma, sca
  le_pow);
        double var4 = variance_from_sample(sample+2, samples, base_sigma, sca
  le_pow);

        //Take a sample
        sampler.set_variance(var2);
        sampler.next(DataForMCMC::get_rng());

        //Compute the SSD. This is fixed regardless of sigma.
        double err_sum=0;
        for(unsigned int frame=0; frame < nframes; frame++)
            for(unsigned int pixel=0; pixel < npixels; pixel++)
                err_sum -= sq(pixel_intensities[frame][pixel] - sampler.sampl
  e_intensities()[frame][pixel]);

        //Compute the derivative using a five point stencil
        //This could be done in a better but less clear way
        double e1 = err_sum / (2*var1) - npixels*nframes*::log(2*M_PI*var1)/2
```

```
      ;
            double e2 = err_sum / (2*var2) - npixels*nframes*::log(2*M_PI*var2)/2
      ;
            double e3 = err_sum / (2*var3) - npixels*nframes*::log(2*M_PI*var3)/2
      ;
            double e4 = err_sum / (2*var4) - npixels*nframes*::log(2*M_PI*var4)/2
      ;
            sum += (-e1 + 8*e2 - 8*e3 + e4)/12;

            ksum.add(-e1/12);
            ksum.add(8*e2/12);
            ksum.add(-8*e3/12);
            ksum.add(e4/12);
        }

        double log_final = (log(variance_from_sample(samples, samples, base_sigma
      , scale_pow)*2*M_PI)/2) * npixels * nframes;

        double priors=0;
        for(unsigned int i=0; i < nspots; i++)
        {
            priors += log_log_normal(spots[i*4+0], mu_brightness,
      sigma_brightness);
            priors += log_log_normal(spots[i*4+1], mu_blur, sigma_blur);
        }

        /*cout << "Thermo:\n";
        cout << "sum: " << sum -log_final << endl;
        cout << "kah: " << ksum.sum -log_final << endl;
        cout << "priors: " << priors + sum -log_final << endl;
        cout << "   kah: " << priors + ksum.sum -log_final << endl;
*/
        //cout << log_final << endl;
        //cout << sum + log_final << endl;


        sampler.set_variance(variance);
        return -(sum+priors - log_final);
    }
```

10.22.3.3 double NegativeFreeEnergy::operator() (const Vector<> & *spots*) const [inline]

Estimate free energy using the Slow Growth Thermodynamic Integration method given in Probalistic Inference Using Markov Chain Monte Carlo Methods, Radford.

M. Neal, 1993 Except using a 5 point stencil instead of forward differenceing in Eq 6.30

Parameters

spots	list of spots

Definition at line 372 of file multispot5.cc.

References Kahan::add(), SampledMultispot::compute_spot_intensity(), DataForMCMC::get_rng(), log_log_normal(), SampledMultispot::GibbsSampler::next(), SampledMultispot::GibbsSampler::sample_intensities(), SampledMultispot::GibbsSampler::set_variance(), spots_to_vector(), and

sq().

```
    {
        double base_sigma = sqrt(variance); // should be 1
        double scale_pow = 100.0;

        const unsigned int nspots  = spots.size()/4;
        const unsigned int nframes = pixel_intensities.size();
        const unsigned int npixels = pixels.size();
        assert(spots.size() %4 == 0);

        //Compute the intensities and derivatives for all spot
        vector<vector<double> > spot_intensity; //[spot][pixel]
        for(unsigned int i=0; i < nspots; i++)
            spot_intensity.push_back(compute_spot_intensity(pixels, spots.slice<D
      ynamic,4>(i*4,4)));

        GibbsSampler sampler(pixel_intensities, spot_intensity, spots_to_vector(s
      pots), A, pi, variance, sample_iterations);

        double sum = 0;
        Kahan ksum;
        for(int sample=0; sample < samples; sample++)
        {
            //Compute the positions of the surrounding steps
            double var1 = variance_from_sample(sample-2, samples, base_sigma, sca
      le_pow);
            double var2 = variance_from_sample(sample-1, samples, base_sigma, sca
      le_pow);
            double var3 = variance_from_sample(sample+1, samples, base_sigma, sca
      le_pow);
            double var4 = variance_from_sample(sample+2, samples, base_sigma, sca
      le_pow);

            //Take a sample
            sampler.set_variance(var2);
            sampler.next(DataForMCMC::get_rng());

            //Compute the SSD. This is fixed regardless of sigma.
            double err_sum=0;
            for(unsigned int frame=0; frame < nframes; frame++)
                for(unsigned int pixel=0; pixel < npixels; pixel++)
                    err_sum -= sq(pixel_intensities[frame][pixel] - sampler.sampl
      e_intensities()[frame][pixel]);

            //Compute the derivative using a five point stencil
            //This could be done in a better but less clear way
            double e1 = err_sum / (2*var1) - npixels*nframes*::log(2*M_PI*var1)/2
      ;
            double e2 = err_sum / (2*var2) - npixels*nframes*::log(2*M_PI*var2)/2
      ;
            double e3 = err_sum / (2*var3) - npixels*nframes*::log(2*M_PI*var3)/2
      ;
            double e4 = err_sum / (2*var4) - npixels*nframes*::log(2*M_PI*var4)/2
      ;
            sum += (-e1 + 8*e2 - 8*e3 + e4)/12;

            ksum.add(-e1/12);
            ksum.add(8*e2/12);
            ksum.add(-8*e3/12);
            ksum.add(e4/12);
        }
```

```
        double log_final = (log(variance_from_sample(samples, samples, base_sigma
      , scale_pow)*2*M_PI)/2) * npixels * nframes;

        double priors=0;
        for(unsigned int i=0; i < nspots; i++)
        {
            priors += log_log_normal(spots[i*4+0], mu_brightness,
      sigma_brightness);
            priors += log_log_normal(spots[i*4+1], mu_blur, sigma_blur);
        }

        /*cout << "Thermo:\n";
        cout << "sum: " << sum -log_final << endl;
        cout << "kah: " << ksum.sum -log_final << endl;
        cout << "priors: " << priors + sum -log_final << endl;
        cout << "   kah: " << priors + ksum.sum -log_final << endl;
*/
        //cout << log_final << endl;
        //cout << sum + log_final << endl;


        sampler.set_variance(variance);
        return -(sum+priors - log_final);
    }
```

10.22.3.4 MT19937& DataForMCMC::get_rng ( ) const [inline, inherited]

Definition at line 189 of file multispot5.cc.

Referenced by compute_with_mask(), and operator()().

```
    {
        return rng;
    }
```

10.22.4 Member Data Documentation

10.22.4.1 const vector<ImageRef>& DataForMCMC::pixels [protected, inherited]

Definition at line 179 of file multispot5.cc.

10.22.4.2 const vector<vector<double> >& DataForMCMC::pixel_intensities [protected, inherited]

Definition at line 180 of file multispot5.cc.

10.22.4.3 const double DataForMCMC::mu_brightness [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.22.4.4 const double DataForMCMC::sigma_brightness [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.22.4.5 const double DataForMCMC::mu_blur [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.22.4.6 const double DataForMCMC::sigma_blur [protected, inherited]

Definition at line 181 of file multispot5.cc.

10.22.4.7 const double DataForMCMC::variance [protected, inherited]

Definition at line 182 of file multispot5.cc.

10.22.4.8 const int DataForMCMC::samples [protected, inherited]

Definition at line 183 of file multispot5.cc.

10.22.4.9 const int DataForMCMC::sample_iterations [protected, inherited]

Definition at line 183 of file multispot5.cc.

10.22.4.10 const Matrix<3> DataForMCMC::A [protected, inherited]

Definition at line 184 of file multispot5.cc.

10.22.4.11 const Vector<3> DataForMCMC::pi [protected, inherited]

Definition at line 185 of file multispot5.cc.

10.22.4.12 MT19937& DataForMCMC::rng [protected, inherited]

Definition at line 186 of file multispot5.cc.

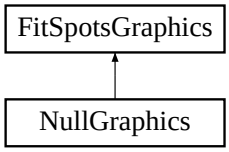
The documentation for this class was generated from the following file:

- multispot5.cc

10.23 NullGraphics Class Reference

Graphics class which does absoloutely nothing.

Inheritance diagram for NullGraphics:



Public Member Functions

- virtual void init (CVD::ImageRef)
- virtual void draw_krap (const std::vector< TooN::Vector< 4 > > &, const CVD::Image< CVD::byte > &, const BBox &, int, TooN::Vector< 4 >)
- virtual void swap ()
- virtual void draw_pixels (const std::vector< CVD::ImageRef > &, float, float, float, float)
- virtual void draw_bbox (const BBox &)
- virtual void glDrawCross (const TooN::Vector< 2 > &, int)
- virtual ~NullGraphics ()

10.23.1 Detailed Description

Graphics class which does absoloutely nothing.

Definition at line 75 of file multispot5.cc.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 virtual NullGraphics::~NullGraphics ( ) [inline, virtual]

Definition at line 84 of file multispot5.cc.

```
{}
```

10.23.3 Member Function Documentation

10.23.3.1 virtual void NullGraphics::init (CVD::ImageRef *size*) [inline, virtual]

Initialize graphics.

Parameters

size	Size of window for display

Implements FitSpotsGraphics.

Definition at line 78 of file multispot5.cc.

```
{}
```

10.23.3.2 virtual void NullGraphics::draw_krap (const std::vector< TooN::Vector< 4 > > & *spots*, const CVD::Image< CVD::byte > & *im*, const BBox & *box*, int *N*, TooN::Vector< 4 > *s*) [inline, virtual]

Draw a bunch of stuff.

Parameters

spots	List of spots to draw
im	Background image
box	Bounding box of region
N	Spot to highlight
s	Extra spot to draw as a cross

Implements FitSpotsGraphics.

Definition at line 79 of file multispot5.cc.

```
{}
```

10.23.3.3 virtual void NullGraphics::swap ( ) [inline, virtual]

Swap buffers if double buffered.

Implements FitSpotsGraphics.

Definition at line 80 of file multispot5.cc.

```
{}
```

10.23.3.4 virtual void NullGraphics::draw_pixels (const std::vector< CVD::ImageRef > & *pix*, float *r*, float *g*, float *b*, float *a*) [inline, virtual]

Draw the pixel mask in an (r,g,b,a) tuple colour.

Parameters

pix	mask
r	red
g	green
b	blue
a	alpha

Implements FitSpotsGraphics.

Definition at line 81 of file multispot5.cc.

```
{}
```

10.23.3.5 virtual void NullGraphics::draw_bbox (const BBox & *bbox*) [inline, virtual]

Draw a bounding box.

Parameters

bbox	Box corners

Implements FitSpotsGraphics.

Definition at line 82 of file multispot5.cc.

```
{}
```

10.23.3.6 virtual void NullGraphics::glDrawCross (const TooN::Vector< 2 > & *p*, int *size*) [inline, virtual]

Draw a cross.

Parameters

p	Position of cross
size	Size to draw cross

Implements FitSpotsGraphics.

Definition at line 83 of file multispot5.cc.

```
{}
```

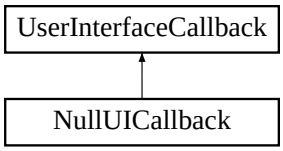
The documentation for this class was generated from the following file:

- multispot5.cc

10.24 NullUICallback Class Reference

User interface callback class which does nothing.

Inheritance diagram for NullUICallback:



Private Member Functions

- void per_spot (int, int, int, int)
- void per_modification (int, int, int)
- void per_pass (int, int, const std::vector< TooN::Vector< 4 > > &)
- void perhaps_stop ()

10.24.1 Detailed Description

User interface callback class which does nothing.

Definition at line 52 of file multispot5.cc.

10.24.2 Member Function Documentation

10.24.2.1 void NullUICallback::per_spot (int *iteration*, int *pass*, int *spot_num*, int *total_spots*) [inline, private, virtual]

This function is called once per spot in each pass.

The idea is to provide a display along the lines of: Iteration #1 optimizing #2% complete

Parameters

iteration	Iteration number

pass	Pass number
spot_num	Spot currently being optimized
total_spots	Total number of spots to be optimized

Implements UserInterfaceCallback.

Definition at line 54 of file multispot5.cc.

```
{};
```

10.24.2.2 void NullUICallback::per_modification (int *iteration*, int *spot_num*, int *total_spots*) [inline, private, virtual]

This function is called once per spot in the modification phase.

The idea is to provide a display along the lines of: Iteration #1 modifying #2% complete

Parameters

iteration	Iteration number
spot_num	Spot currently being optimized
total_spots	Total number of spots to be optimized

Implements UserInterfaceCallback.

Definition at line 55 of file multispot5.cc.

```
{};
```

10.24.2.3 void NullUICallback::per_pass (int *iteration*, int *pass*, const std::vector< TooN::Vector< 4 > > & *spots*) [inline, private, virtual]

This function is called once each time PASS data is outputted.

It will allow the GUI to build up a reconstruction.

Parameters

iteration	Iteration number
pass	Pass number
spots	Data to be reconstructed

Implements UserInterfaceCallback.

Definition at line 56 of file multispot5.cc.

```
{};
```

10.24.2.4 void NullUICallback::perhaps_stop ( ) [inline, private, virtual]

The user wishes to issue a stop instruction to the program (perhaps done via an asynchronus call to an instance of of UserInterfaceCallback).

This function is called as often as possible and will throw UserIssuedStop when the condition is met.

Implements UserInterfaceCallback.

Definition at line 57 of file multispot5.cc.

```
{};
```

The documentation for this class was generated from the following file:

- multispot5.cc

10.25 MT19937::ParseError Struct Reference

Null struct thrown if attempting to load state from stream yields a parse error.

`#include <mt19937.h>`

10.25.1 Detailed Description

Null struct thrown if attempting to load state from stream yields a parse error.

Definition at line 17 of file mt19937.h.

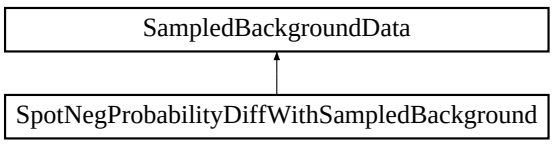
The documentation for this struct was generated from the following file:

- mt19937.h

10.26 SampledBackgroundData Struct Reference

Closure holding image data generated using samples drawn from the model.

Inheritance diagram for SampledBackgroundData:



Public Member Functions

- SampledBackgroundData (const vector< vector< vector< double > > > &sample_intensities_without_spot_, const vector< vector< double > > &pixel_intensities_, const vector< ImageRef > pixels_, double mu_brightness_, double sigma_brightness_, double mu_blur_, double sigma_blur_, const Matrix< 3 > A_, const Vector< 3 > pi_, double variance_)

Public Attributes

- const vector< vector< vector< double > > > & sample_intensities_without_spot
- const vector< vector< double > > & pixel_intensities
- const vector< ImageRef > pixels
- double mu_brightness
- double sigma_brightness
- double mu_blur
- double sigma_blur
- const Matrix< 3 > A
- const Vector< 3 > pi
- double variance
- const vector< int > O

10.26.1 Detailed Description

Closure holding image data generated using samples drawn from the model.

NB this is used with one spot removed (i.e. set to dark). As a result, the data is treated as the background when considering that spot in isolation. See FitSpots for naming of variables.

Definition at line 484 of file multispot5.cc.

10.26.2 Constructor & Destructor Documentation

10.26.2.1 SampledBackgroundData::SampledBackgroundData (const vector< vector< vector< double > > > & *sample_intensities_without_spot_*, const vector< vector< double > > & *pixel_intensities_*, const vector< ImageRef > *pixels_*, double *mu_brightness_*, double *sigma_brightness_*, double *mu_blur_*, double *sigma_blur_*, const Matrix< 3 > *A_*, const Vector< 3 > *pi_*, double *variance_*) [inline]

Definition at line 497 of file multispot5.cc.

```
:sample_intensities_without_spot(sample_intensities_without_spot_),
    pixel_intensities(pixel_intensities_),
    pixels(pixels_),
    mu_brightness(mu_brightness_),
    sigma_brightness(sigma_brightness_),
```

```
        mu_blur(mu_blur_),
        sigma_blur(sigma_blur_),
        A(A_),
        pi(pi_),
        variance(variance_),
        O(sequence(pixel_intensities.size()))
    {
    }
```

10.26.3 Member Data Documentation

10.26.3.1 const vector<vector<vector<double> > >& SampledBackgroundData::sample_intensities_without_spot

Definition at line 486 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.2 const vector<vector<double> >& SampledBackgroundData::pixel_-intensities

Definition at line 487 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.3 const vector<ImageRef> SampledBackgroundData::pixels

Definition at line 488 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.4 double SampledBackgroundData::mu_brightness

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.5 double SampledBackgroundData::sigma_brightness

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-

FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.6 double SampledBackgroundData::mu_blur

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.7 double SampledBackgroundData::sigma_blur

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.8 const Matrix<3> SampledBackgroundData::A

Definition at line 491 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.9 const Vector<3> SampledBackgroundData::pi

Definition at line 492 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.10 double SampledBackgroundData::variance

Definition at line 493 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.26.3.11 const vector<int> SampledBackgroundData::O

Definition at line 495 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

The documentation for this struct was generated from the following file:

- multispot5.cc

10.27 FloatSliderWithBox::SliderChanged Class Reference

Public Member Functions

- void stateChanged (ChangeEvent e)

Package Functions

- SliderChanged (FloatSliderWithBox f_)

Package Attributes

- FloatSliderWithBox f

10.27.1 Detailed Description

Definition at line 803 of file three_B.java.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 FloatSliderWithBox::SliderChanged::SliderChanged (FloatSliderWithBox *f_*)

`[inline, package]`

Definition at line 806 of file three_B.java.

References f.

```
{
    f = f_;
}
```

10.27.3 Member Function Documentation

10.27.3.1 void FloatSliderWithBox::SliderChanged::stateChanged (ChangeEvent *e*)

`[inline]`

Definition at line 811 of file three_B.java.

References f, FloatSliderWithBox::get_value_from_slider(), and FloatSliderWithBox::setValue().

```
{
    f.setValue(f.get_value_from_slider());
}
```

10.27.4 Member Data Documentation

10.27.4.1 FloatSliderWithBox FloatSliderWithBox::SliderChanged::f [package]

Definition at line 805 of file three_B.java.

Referenced by SliderChanged(), and stateChanged().

The documentation for this class was generated from the following file:

- three_B.java

10.28 SomethingChanges Class Reference

Listener class which triggers a complete redraw.

Public Member Functions

- SomethingChanges (EControlPanel c_)
- void stateChanged (ChangeEvent e)

Private Attributes

- EControlPanel c

10.28.1 Detailed Description

Listener class which triggers a complete redraw.

Since all redraws are pretty much equal, there is no need to distinguish them.

Definition at line 666 of file three_B.java.

10.28.2 Constructor & Destructor Documentation

10.28.2.1 SomethingChanges::SomethingChanges (EControlPanel *c_*) [inline]

Definition at line 669 of file three_B.java.

References c.

```
{
    c = c_;
}
```

10.28.3 Member Function Documentation

10.28.3.1 void SomethingChanges::stateChanged (ChangeEvent *e*) [inline]

Definition at line 675 of file three_B.java.

References c, and EControlPanel::send_update_canvas_event().

```
{
    c.send_update_canvas_event();
}
```

10.28.4 Member Data Documentation

10.28.4.1 EControlPanel SomethingChanges::c [private]

Definition at line 668 of file three_B.java.

Referenced by SomethingChanges(), and stateChanged().

The documentation for this class was generated from the following file:

- three_B.java

10.29 SPair Class Reference

Utility class to hold a pair of strings.

Public Attributes

- String a

Package Attributes

- String b

10.29.1 Detailed Description

Utility class to hold a pair of strings.

Definition at line 38 of file three_B.java.

10.29.2 Member Data Documentation

10.29.2.1 String SPair::a

Definition at line 40 of file three_B.java.

Referenced by Util::getFileName(), and three_B::run().

10.29.2.2 String SPair::b [package]

Definition at line 40 of file three_B.java.

Referenced by Util::getFileName(), and three_B::run().

The documentation for this class was generated from the following file:

- three_B.java

10.30 Spot Class Reference

Basic spot class, simply contains coordinates.

Package Functions

- Spot ()
- Spot (double xx, double yy)

Package Attributes

- double x
- double y

10.30.1 Detailed Description

Basic spot class, simply contains coordinates.

Definition at line 648 of file three_B.java.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 Spot::Spot ( ) [inline, package]

Definition at line 651 of file three_B.java.

```
{
}
```

10.30.2.2 Spot::Spot (double *xx*, double *yy*) [inline, package]

Definition at line 655 of file three_B.java.

References x, and y.

```
{
    x=xx;
    y=yy;
}
```

10.30.3 Member Data Documentation

10.30.3.1 double Spot::x [package]

Definition at line 650 of file three_B.java.

Referenced by ThreeBLoader::run(), ThreeBRunner::send_new_points(), and Spot().

10.30.3.2 double Spot::y [package]

Definition at line 650 of file three_B.java.

Referenced by ThreeBLoader::run(), ThreeBRunner::send_new_points(), and Spot().

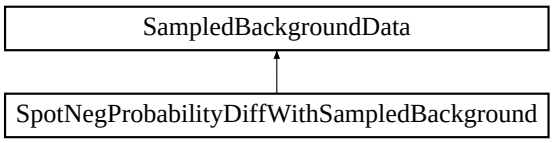
The documentation for this class was generated from the following file:

- three_B.java

10.31 SpotNegProbabilityDiffWithSampledBackground Struct Reference

Compute the derivative of the negative log probability with respect to the parameters of one spot, given some samples of the other spots.

Inheritance diagram for SpotNegProbabilityDiffWithSampledBackground:



Public Member Functions

- SpotNegProbabilityDiffWithSampledBackground (const SampledBackgroundData &d)
- Vector< 4 > operator() (const Vector< 4 > &spot) const

Public Attributes

- const vector< vector< vector< double > > > & sample_intensities_without_spot
- const vector< vector< double > > & pixel_intensities
- const vector< ImageRef > pixels
- double mu_brightness
- double sigma_brightness
- double mu_blur
- double sigma_blur
- const Matrix< 3 > A
- const Vector< 3 > pi
- double variance
- const vector< int > O

10.31.1 Detailed Description

Compute the derivative of the negative log probability with respect to the parameters of one spot, given some samples of the other spots.

Definition at line 567 of file multispot5.cc.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 SpotNegProbabilityDiffWithSampledBackground::SpotNegProbabilityDiffWithSampledBackground (const SampledBackgroundData & *d*) [inline]

Parameters

d	Necessary data for construction

Definition at line 570 of file multispot5.cc.

```
:SampledBackgroundData(d)
{
}
```

10.31.3 Member Function Documentation

10.31.3.1 Vector<4> SpotNegProbabilityDiffWithSampledBackground::operator() (const Vector< 4 > & *spot*) const [inline]

Compute the probability of spot.

Parameters

spot	Spot position

Definition at line 577 of file multispot5.cc.

References SampledMultispot::compute_spot_intensity_derivatives(), diff_log_log_normal(), and forward_algorithm_deriv().

```
{
    if(spot[0] <= 0 || spot[1] <= 0)
        return Ones * std::numeric_limits<double>::quiet_NaN();

    vector<pair<double, Vector<4> > > spot_intensities =
  compute_spot_intensity_derivatives(pixels, spot);

    Vector<4> sum_diff_log = Zeros;

    for(unsigned int s=0; s < sample_intensities_without_spot.size(); s++)
    {
        SpotWithBackground B(sample_intensities_without_spot[s], spot_intensi
  ties, pixel_intensities, variance);

        pair<double, Vector<4> > r = forward_algorithm_deriv(A, pi, B, O);

        sum_diff_log += r.second;
    }

    Vector<4> diff_log = sum_diff_log / sample_intensities_without_spot.size(
  );

    //Compute the log probability of the prior
    Vector<4> logprior_deriv = makeVector(diff_log_log_normal(spot[0],
  mu_brightness, sigma_brightness),
```

```
                                                diff_log_log_normal(spot[1],
    mu_blur, sigma_blur), 0, 0);

      return -(diff_log + logprior_deriv);
  }
```

10.31.4 Member Data Documentation

10.31.4.1 const vector<vector<vector<double> > >& SampledBackgroundData::sample_intensities_without_spot [inherited]

Definition at line 486 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.2 const vector<vector<double> >& SampledBackgroundData::pixel_-intensities [inherited]

Definition at line 487 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.3 const vector<ImageRef> SampledBackgroundData::pixels [inherited]

Definition at line 488 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.4 double SampledBackgroundData::mu_brightness [inherited]

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.5 double SampledBackgroundData::sigma_brightness [inherited]

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.6 double SampledBackgroundData::mu_blur [inherited]

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.7 double SampledBackgroundData::sigma_blur [inherited]

Definition at line 490 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.8 const Matrix<3> SampledBackgroundData::A [inherited]

Definition at line 491 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.9 const Vector<3> SampledBackgroundData::pi [inherited]

Definition at line 492 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.10 double SampledBackgroundData::variance [inherited]

Definition at line 493 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

10.31.4.11 const vector<int> SampledBackgroundData::O [inherited]

Definition at line 495 of file multispot5.cc.

Referenced by sampled_background_spot_hessian2(), sampled_background_spot_hessian_-FAKE(), and sampled_background_spot_hessian_ffbs().

The documentation for this struct was generated from the following file:

- multispot5.cc

10.32 SampledMultispot::SpotWithBackgroundMasked Struct Reference

This class compute the log-diff-hess probability of a spot, given an image patch and background due to existing spots.

`#include <sampled_multispot.h>`

Public Member Functions

- double get_val (double d)
- Vector< 4 > get_diff (double)
- Matrix< 4 > get_hess (double)
- double get_val (const pair< double, Vector< 4 > > &d)
- Vector< 4 > get_diff (const pair< double, Vector< 4 > > &d)
- Matrix< 4 > get_hess (const pair< double, Vector< 4 > > &)
- double get_val (const tuple< double, Vector< 4 >, Matrix< 4 > > &d)
- Vector< 4 > get_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > &d)
- Matrix< 4 > get_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > &d)
- template<class C >
  bool type_has_hess (const C &)
- bool type_has_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > &)
- template<class C >
  bool type_has_diff (const C &)
- bool type_has_diff (const pair< double, Vector< 4 > > &)
- bool type_has_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > &)
- template<class Input >
  SpotWithBackgroundMasked (const vector< vector< double > > &sample_-intensities, const SWBG_SPOT_INTENSITIES &spot_intensities, const vector< vector< double > > &pixel_intensities, const double variance, const vector< int > &mask)
- double log (int state, int obs) const
- Vector< 4 > diff_log (int state, int obs) const
- Matrix< 4 > hess_log (int state, int obs) const
- double get_val (double d)
- Vector< 4 > get_diff (double)
- Matrix< 4 > get_hess (double)
- double get_val (const pair< double, Vector< 4 > > &d)
- Vector< 4 > get_diff (const pair< double, Vector< 4 > > &d)
- Matrix< 4 > get_hess (const pair< double, Vector< 4 > > &)
- double get_val (const tuple< double, Vector< 4 >, Matrix< 4 > > &d)
- Vector< 4 > get_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > &d)
- Matrix< 4 > get_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > &d)
- template<class C >
  bool type_has_hess (const C &)

- bool type_has_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > &)
- template<class C >
  bool type_has_diff (const C &)
- bool type_has_diff (const pair< double, Vector< 4 > > &)
- bool type_has_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > &)
- template<class Input >
  SpotWithBackgroundMasked (const vector< vector< double > > &sample_intensities, const SWBG_SPOT_INTENSITIES &spot_intensities, const vector< vector< double > > &pixel_intensities, const double variance, const vector< int > &mask)
- double log (int state, int obs) const
- Vector< 4 > diff_log (int state, int obs) const
- Matrix< 4 > hess_log (int state, int obs) const

Public Attributes

- vector< pair< double, double > > log_prob
- vector< Vector< 4 > > diff_log_prob
- vector< Matrix< 4 > > hess_log_prob

Static Public Attributes

- static const int NumParameters = 4

10.32.1 Detailed Description

This class compute the log-diff-hess probability of a spot, given an image patch and background due to existing spots.

Definition at line 5 of file sampled_multispot.h.

10.32.2 Constructor & Destructor Documentation

10.32.2.1 template<class Input > SampledMultispot::SpotWithBackgroundMasked::SpotWithBackgroundMasked (const vector< vector< double > > & *sample_intensities*, const SWBG_SPOT_INTENSITIES & *spot_intensities*, const vector< vector< double > > & *pixel_intensities*, const double *variance*, const vector< int > & *mask*) [inline]

Definition at line 40 of file sampled_multispot.h.

```
{
    return i;
}

inline double intensity(const pair<double, Vector<4> >& i)
```

```
{
    return i.first;
}


//Add and remove a spot over the entire region
template<class T>
void remove_spot(vector<vector<double> >& current_sample_intensities, const vecto
      r<T>& spot_intensities, const vector<State>& spot_sample)
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < spot_intensities.size(); p++)
                current_sample_intensities[frame][p] -= intensity(spot_intensitie
      s[p]);
}

template<class T>
void add_spot(vector<vector<double> >& current_sample_intensities, const vector<T
      >& spot_intensities, const vector<State>& spot_sample)
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < spot_intensities.size(); p++)
                current_sample_intensities[frame][p] += intensity(spot_intensitie
      s[p]);
}


//Add and remove a spot only over a mask.
template<class T>
void remove_spot(vector<vector<double> >& current_sample_intensities, const vecto
      r<T>& spot_intensities, const vector<State>& spot_sample, const vector<int>& mask
      )
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] -= intensity(spot_inte
      nsities[mask[p]]);
}

template<class T>
void add_spot(vector<vector<double> >& current_sample_intensities, const vector<T
      >& spot_intensities, const vector<State>& spot_sample, const vector<int>& mask)
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] += intensity(spot_inte
      nsities[mask[p]]);
}


//Add and remove a drifty spot only over a mask.
template<class T>
void remove_spot(vector<vector<double> >& current_sample_intensities, const vecto
      r<vector<T> > & spot_intensities, const vector<State>& spot_sample, const vector<
```

```
      int>& mask)
{
    const int steps = spot_intensities.size();
    const int frames = current_sample_intensities.size();

    for(int frame=0; frame < frames; frame++)
    {
        int s = frame * steps / frames;

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] -= intensity(spot_inte
      nsities[s][mask[p]]);
    }
}

template<class T>
void add_spot(vector<vector<double> >& current_sample_intensities, const vector<v
      ector<T> >& spot_intensities, const vector<State>& spot_sample, const vector<int>
      & mask)
{
    const int steps = spot_intensities.size();
    const int frames = current_sample_intensities.size();

    for(int frame=0; frame < frames; frame++)
    {
        int s = frame * steps / frames;

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] += intensity(spot_inte
      nsities[s][mask[p]]);
    }
}



//Compute the spot intensity for a given spot at each pixel
inline vector<double> compute_spot_intensity(const vector<ImageRef>& pixels, cons
      t Vector<4>& params)
```

10.32.2.2 template<class Input > SampledMultispot::SpotWithBackgroundMasked::SpotWithBackgroundMasked (const vector< vector< double > > & *sample_intensities*, const SWBG_SPOT_INTENSITIES & *spot_intensities*, const vector< vector< double > > & *pixel_intensities*, const double *variance*, const vector< int > & *mask*) [inline]

Definition at line 40 of file sampled_multispot.h.

```
{
    return i;
}

inline double intensity(const pair<double, Vector<4> >& i)
{
    return i.first;
}
```

```
//Add and remove a spot over the entire region
template<class T>
void remove_spot(vector<vector<double> >& current_sample_intensities, const vecto
      r<T>& spot_intensities, const vector<State>& spot_sample)
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < spot_intensities.size(); p++)
                current_sample_intensities[frame][p] -= intensity(spot_intensitie
      s[p]);
}

template<class T>
void add_spot(vector<vector<double> >& current_sample_intensities, const vector<T
      >& spot_intensities, const vector<State>& spot_sample)
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < spot_intensities.size(); p++)
                current_sample_intensities[frame][p] += intensity(spot_intensitie
      s[p]);
}


//Add and remove a spot only over a mask.
template<class T>
void remove_spot(vector<vector<double> >& current_sample_intensities, const vecto
      r<T>& spot_intensities, const vector<State>& spot_sample, const vector<int>& mask
      )
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] -= intensity(spot_inte
      nsities[mask[p]]);
}

template<class T>
void add_spot(vector<vector<double> >& current_sample_intensities, const vector<T
      >& spot_intensities, const vector<State>& spot_sample, const vector<int>& mask)
{
    for(unsigned int frame=0; frame < current_sample_intensities.size(); frame++)

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] += intensity(spot_inte
      nsities[mask[p]]);
}


//Add and remove a drifty spot only over a mask.
template<class T>
void remove_spot(vector<vector<double> >& current_sample_intensities, const vecto
      r<vector<T> > & spot_intensities, const vector<State>& spot_sample, const vector<
      int>& mask)
{
    const int steps = spot_intensities.size();
    const int frames = current_sample_intensities.size();
```

```
    for(int frame=0; frame < frames; frame++)
    {
        int s = frame * steps / frames;

        if(spot_sample[frame] == 0) //Spot is on, so remove it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] -= intensity(spot_inte
      nsities[s][mask[p]]);
    }
}

template<class T>
void add_spot(vector<vector<double> >& current_sample_intensities, const vector<v
      ector<T> >& spot_intensities, const vector<State>& spot_sample, const vector<int>
      & mask)
{
    const int steps = spot_intensities.size();
    const int frames = current_sample_intensities.size();

    for(int frame=0; frame < frames; frame++)
    {
        int s = frame * steps / frames;

        if(spot_sample[frame] == 0) //Spot is on, so add it
            for(unsigned int p=0; p < mask.size(); p++)
                current_sample_intensities[frame][mask[p]] += intensity(spot_inte
      nsities[s][mask[p]]);
    }
}



//Compute the spot intensity for a given spot at each pixel
inline vector<double> compute_spot_intensity(const vector<ImageRef>& pixels, cons
      t Vector<4>& params)
```

10.32.3 Member Function Documentation

10.32.3.1 double SampledMultispot::SpotWithBackgroundMasked::get_val (double *d*) [inline]

Definition at line 13 of file sampled_multispot.h.

```
{
```

10.32.3.2 Vector<4> SampledMultispot::SpotWithBackgroundMasked::get_diff (double) [inline]

Definition at line 14 of file sampled_multispot.h.

```
{
```

10.32.3.3 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::get_hess (double) [inline]

Definition at line 15 of file sampled_multispot.h.

10.32.3.4 double SampledMultispot::SpotWithBackgroundMasked::get_val (const pair< double, Vector< 4 > > & *d*) [inline]

Definition at line 17 of file sampled_multispot.h.

10.32.3.5 Vector<4> SampledMultispot::SpotWithBackgroundMasked::get_diff (const pair< double, Vector< 4 > > & *d*) [inline]

Definition at line 18 of file sampled_multispot.h.

10.32.3.6 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::get_hess (const pair< double, Vector< 4 > > &) [inline]

Definition at line 19 of file sampled_multispot.h.

{

10.32.3.7 double SampledMultispot::SpotWithBackgroundMasked::get_val (const tuple< double, Vector< 4 >, Matrix< 4 > > & *d*) [inline]

Definition at line 21 of file sampled_multispot.h.

{

10.32.3.8 Vector<4> SampledMultispot::SpotWithBackgroundMasked::get_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > & *d*) [inline]

Definition at line 22 of file sampled_multispot.h.

{

10.32.3.9 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::get_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > & *d*) [inline]

Definition at line 23 of file sampled_multispot.h.

{

10.32.3.10 template<class C > bool SampledMultispot::SpotWithBackgroundMasked::type_has_hess (const C &) [inline]

Definition at line 25 of file sampled_multispot.h.

{

10.32.3.11 bool SampledMultispot::SpotWithBackgroundMasked::type_has_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > &) [inline]

Definition at line 26 of file sampled_multispot.h.

{

10.32.3.12 template<class C > bool SampledMultispot::SpotWithBackgroundMasked::type_has_diff (const C &) [inline]

Definition at line 28 of file sampled_multispot.h.

{

10.32.3.13 bool SampledMultispot::SpotWithBackgroundMasked::type_has_diff (const pair< double, Vector< 4 > > &) [inline]

Definition at line 29 of file sampled_multispot.h.

{

10.32.3.14 bool SampledMultispot::SpotWithBackgroundMasked::type_has_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > &) [inline]

Definition at line 30 of file sampled_multispot.h.

{

10.32.3.15 double SampledMultispot::SpotWithBackgroundMasked::log (int *state,* int *obs*) const [inline]

Definition at line 135 of file sampled_multispot.h.

```
{
```

10.32.3.16 Vector<4> SampledMultispot::SpotWithBackgroundMasked::diff_log (int *state,* int *obs*) const [inline]

Definition at line 146 of file sampled_multispot.h.

```
{
    vector<tuple<double, Vector<4>, Matrix<4> > > hessian(pixels.size());
```

10.32.3.17 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::hess_log (int *state,* int *obs*) const [inline]

Definition at line 157 of file sampled_multispot.h.

```
{
```

10.32.3.18 double SampledMultispot::SpotWithBackgroundMasked::get_val (double *d*) [inline]

Definition at line 13 of file sampled_multispot.h.

```
{
```

10.32.3.19 Vector<4> SampledMultispot::SpotWithBackgroundMasked::get_diff (double) [inline]

Definition at line 14 of file sampled_multispot.h.

```
{
```

10.32.3.20 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::get_hess (double) [inline]

Definition at line 15 of file sampled_multispot.h.

10.32.3.21 double SampledMultispot::SpotWithBackgroundMasked::get_val (const pair< double, Vector< 4 > > & *d*) [inline]

Definition at line 17 of file sampled_multispot.h.

10.32.3.22 Vector<4> SampledMultispot::SpotWithBackgroundMasked::get_diff (const pair< double, Vector< 4 > > & *d*) [inline]

Definition at line 18 of file sampled_multispot.h.

10.32.3.23 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::get_hess (const pair< double, Vector< 4 > > &) [inline]

Definition at line 19 of file sampled_multispot.h.

{

10.32.3.24 double SampledMultispot::SpotWithBackgroundMasked::get_val (const tuple< double, Vector< 4 >, Matrix< 4 > > & *d*) [inline]

Definition at line 21 of file sampled_multispot.h.

{

10.32.3.25 Vector<4> SampledMultispot::SpotWithBackgroundMasked::get_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > & *d*) [inline]

Definition at line 22 of file sampled_multispot.h.

{

10.32.3.26 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::get_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > & *d*) [inline]

Definition at line 23 of file sampled_multispot.h.

{

10.32.3.27 template<class C > bool SampledMultispot::SpotWithBackgroundMasked::type_has_hess (const C &) [inline]

Definition at line 25 of file sampled_multispot.h.

{

10.32.3.28 bool SampledMultispot::SpotWithBackgroundMasked::type_has_hess (const tuple< double, Vector< 4 >, Matrix< 4 > > &) [inline]

Definition at line 26 of file sampled_multispot.h.

{

10.32.3.29 template<class C > bool SampledMultispot::SpotWithBackgroundMasked::type_has_diff (const C &) [inline]

Definition at line 28 of file sampled_multispot.h.

{

10.32.3.30 bool SampledMultispot::SpotWithBackgroundMasked::type_has_diff (const pair< double, Vector< 4 > > &) [inline]

Definition at line 29 of file sampled_multispot.h.

{

10.32.3.31 bool SampledMultispot::SpotWithBackgroundMasked::type_has_diff (const tuple< double, Vector< 4 >, Matrix< 4 > > &) [inline]

Definition at line 30 of file sampled_multispot.h.

{

10.32.3.32 double SampledMultispot::SpotWithBackgroundMasked::log (int *state,* int *obs*) const [inline]

Definition at line 135 of file sampled_multispot.h.

{

10.32.3.33 Vector<4> SampledMultispot::SpotWithBackgroundMasked::diff_log (int *state*, int *obs*) const [inline]

Definition at line 146 of file sampled_multispot.h.

```
{
    vector<tuple<double, Vector<4>, Matrix<4> > > hessian(pixels.size());
```

10.32.3.34 Matrix<4> SampledMultispot::SpotWithBackgroundMasked::hess_log (int *state*, int *obs*) const [inline]

Definition at line 157 of file sampled_multispot.h.

```
{
```

10.32.4 Member Data Documentation

10.32.4.1 static const int SampledMultispot::SpotWithBackgroundMasked::NumParameters = 4 [static]

Definition at line 7 of file sampled_multispot.h.

10.32.4.2 vector< pair< double, double > > SampledMultispot::SpotWithBackgroundMasked::log_prob

Definition at line 9 of file sampled_multispot.h.

10.32.4.3 vector< Vector< 4 > > SampledMultispot::SpotWithBackgroundMasked::diff_log_prob

Definition at line 10 of file sampled_multispot.h.

10.32.4.4 vector< Matrix< 4 > > SampledMultispot::SpotWithBackgroundMasked::hess_log_prob

Definition at line 11 of file sampled_multispot.h.

The documentation for this struct was generated from the following file:

10.33 StateParameters Struct Reference

Internal state (excluding fixed settings) which represents the entire internal state of spot fitting.

`#include <multispot5.h>`

Public Attributes

- std::tr1::shared_ptr< MT19937 > rng
- std::vector< TooN::Vector< 4 > > spots
- int pass
- int iteration
- std::vector< CVD::ImageRef > pixels

10.33.1 Detailed Description

Internal state (excluding fixed settings) which represents the entire internal state of spot fitting.

Used to restart from interruptions.

Definition at line 115 of file multispot5.h.

10.33.2 Member Data Documentation

10.33.2.1 std::tr1::shared_ptr<MT19937> StateParameters::rng

Random number generator state.

Definition at line 116 of file multispot5.h.

Referenced by generate_state_parameters_ye_olde(), and parse_log_file().

10.33.2.2 std::vector<TooN::Vector<4> > StateParameters::spots

Spots positions.

Definition at line 117 of file multispot5.h.

Referenced by generate_state_parameters_ye_olde(), and parse_log_file().

10.33.2.3 int StateParameters::pass

Pass number.

Definition at line 118 of file multispot5.h.

Referenced by generate_state_parameters_ye_olde(), and parse_log_file().

10.33.2.4 int StateParameters::iteration

Iteration number.

Definition at line 119 of file multispot5.h.

Referenced by generate_state_parameters_ye_olde(), and parse_log_file().

10.33.2.5 std::vector<CVD::ImageRef> StateParameters::pixels

Area for analysis.

Definition at line 120 of file multispot5.h.

Referenced by generate_state_parameters_ye_olde(), and parse_log_file().

The documentation for this struct was generated from the following file:

- multispot5.h

10.34 StopButtonListener Class Reference

Stop 3B thread.

Public Member Functions

- StopButtonListener (EControlPanel t_)
- void actionPerformed (ActionEvent e)

Private Attributes

- EControlPanel t

10.34.1 Detailed Description

Stop 3B thread.

Definition at line 856 of file three_B.java.

10.34.2 Constructor & Destructor Documentation

10.34.2.1 StopButtonListener::StopButtonListener (EControlPanel *t_*) [inline]

Definition at line 859 of file three_B.java.

References t.

```
{
    t = t_;
}
```

10.34.3 Member Function Documentation

10.34.3.1 void StopButtonListener::actionPerformed (ActionEvent *e*) [inline]

Definition at line 864 of file three_B.java.

References EControlPanel::issue_stop(), and t.

```
{
    t.issue_stop();
}
```

10.34.4 Member Data Documentation

10.34.4.1 EControlPanel StopButtonListener::t [private]

Definition at line 858 of file three_B.java.

Referenced by actionPerformed(), and StopButtonListener().

The documentation for this class was generated from the following file:

- three_B.java

10.35 FloatSliderWithBox::TextChanged Class Reference

Public Member Functions

- void actionPerformed (ActionEvent e)

Package Functions

- TextChanged (FloatSliderWithBox f_)

Package Attributes

- FloatSliderWithBox f

10.35.1 Detailed Description

Definition at line 817 of file three_B.java.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 FloatSliderWithBox::TextChanged::TextChanged (FloatSliderWithBox *f_*) [inline, package]

Definition at line 820 of file three_B.java.

References f.

```
{
    f = f_;
}
```

10.35.3 Member Function Documentation

10.35.3.1 void FloatSliderWithBox::TextChanged::actionPerformed (ActionEvent *e*) [inline]

Definition at line 825 of file three_B.java.

References f, FloatSliderWithBox::get_value_from_text(), and FloatSliderWithBox::setValue().

```
{
    f.setValue(f.get_value_from_text());
}
```

10.35.4 Member Data Documentation

10.35.4.1 FloatSliderWithBox FloatSliderWithBox::TextChanged::f [package]

Definition at line 819 of file three_B.java.

Referenced by actionPerformed(), and TextChanged().

The documentation for this class was generated from the following file:

- three_B.java

10.36 three_B Class Reference

ImageJ plugin class.

Inherits PlugInFilter.

Public Member Functions

- int setup (String arg_, ImagePlus img)
- void run (ImageProcessor ip)

Package Attributes

- ImagePlus window
- ByteProcessor mask
- String arg

10.36.1 Detailed Description

ImageJ plugin class.

Definition at line 132 of file three_B.java.

10.36.2 Member Function Documentation

10.36.2.1 int three_B::setup (String *arg_*, ImagePlus *img*) [inline]

Definition at line 138 of file three_B.java.

References arg, and window.

```
                                                    {
      window = img;
      arg=arg_;

      return ROI_REQUIRED + SUPPORTS_MASKING + STACK_REQUIRED +NO_CHANGES + DOE
    S_16 + DOES_32 + DOES_8G;
  }
```

10.36.2.2 void three_B::run (ImageProcessor *ip*) [inline]

Definition at line 145 of file three_B.java.

References SPair::a, arg, SPair::b, Util::getFileName(), mask, Util::read(), and window.

```
                                    {


  //Load the config file contents
  Reader cfgstream = new InputStreamReader(getClass().getClassLoader().getR
esourceAsStream("multispot5.cfg"));
  String cfg = Util.read(cfgstream);

  try{
      cfgstream.close();
  }
  catch(IOException close_err){
      Toolkit.getDefaultToolkit().beep();
      ij.IJ.showStatus("Error reading config file.");
      return;
  }

  //Some basic error checking if reading of the config file from
  //the JAR archive fails.
  if(cfg == "")
  {
      Toolkit.getDefaultToolkit().beep();
      ij.IJ.showStatus("Error reading config file.");
      return;
  }


  //The image from getMask() is only the size of the ROI
  //We need it to be congruent with the original image, in order
  //to work with the C++ code.
  mask = new ByteProcessor(ip.getWidth(), ip.getHeight());

  int x = ip.getRoi().x;
  int y = ip.getRoi().y;

  //Rectangular selections do not have a mask set, so we get
  //a null pointer exception when we try to copy.
  //So, we have to manually set the pixels, rather than just
  //copy them
  try{
      mask.copyBits(ip.getMask(), x, y, Blitter.COPY);
  }
  catch(NullPointerException e)
  {
      for(int r=0; r < ip.getRoi().height; r++)
          for(int c=0; c < ip.getRoi().width; c++)
              mask.set(c+x, r+y, 255);
  }

  //Count the number of set pixels in the mask. This is used to
  //warn the user if the number is not within a reasonable range.
  int count=0;
  for(int r=0; r < mask.getHeight(); r++)
      for(int c=0; c < mask.getWidth(); c++)
          if(mask.get(c, r) != 0)
```

```
            count ++;


  //The non-config file parameters are the range of frames to  operate on
  //and the pixel size in nm (the config works in terms of FWHM in pixels).

  //These have to be sepficied whether the basic or advanced dialog is used
.
  int  firstfr;
  int lastfr;
  double pixel_size_in_nm;

  ImageStack s = window.getStack();

  if(arg.equals("advanced"))
  {
      AdvancedDialog ad = new AdvancedDialog(cfg, s.getSize());
      if(!ad.wasOKed())
          return;
      cfg = ad.getTextArea1().getText();


      /*pixel_size_in_nm = ad.getPixelSize();
      firstfr = ad.getFirstFrame();
      lastfr = ad.getLastFrame();*/


      pixel_size_in_nm = ad.getNextNumber();
      firstfr = (int)ad.getNextNumber();
      lastfr = (int)ad.getNextNumber();
  }
  else
  {
      ThreeBDialog gd = new ThreeBDialog(count, mask.getWidth()*mask.getHei
ght(), s.getSize());

      gd.showDialog();

      if(!gd.wasOKed())
          return;

      /*final double fwhm = gd.getFWHM();
      pixel_size_in_nm = gd.getPixelSize();
      final int initial_spots = gd.getSpots();
      firstfr = gd.getFirstFrame();
      lastfr = gd.getLastFrame();*/

      //We have to use getNextNumber, otherwise macro recording does not wo
rk.
      final double fwhm = gd.getNextNumber();
      pixel_size_in_nm = gd.getNextNumber();
      final int initial_spots = (int)gd.getNextNumber();
      firstfr = (int)gd.getNextNumber();
      lastfr = (int)gd.getNextNumber();


      //Compute the parameters of the log-normal prior such that the mode
      //matches the size of the spots.
      final double sigma = (fwhm / pixel_size_in_nm) / (2*Math.sqrt(2*Math.
log(2)));
      final double blur_sigma=0.1;
```

```
        //s = exp(mu-sig^2)
        //ln s = mu - sig^2
        //mu = ln s + sig^2

        final double blur_mu = Math.log(sigma) + blur_sigma*blur_sigma;

        //Initialized from the current time.
        Random rng = new Random();
        //


        //Append stuff to the config file now. This will be parsed later in C
++.
        cfg = cfg + "placement.uniform.num_spots=" + Integer.toString(initial
_spots) + "\n"
                  + "blur.mu="    + Double.toString(blur_mu) + "\n"
                  + "blur.sigma=" + Double.toString(blur_sigma) + "\n"
                  + "seed=" + Integer.toString(rng.nextInt(16777216)) + "\n";


      }

      //Acquire a filename to save moderately safely.
      SPair f = Util.getFileName(window.getTitle());
      final String fname = f.a;
      final String fullname = f.b;

      if(fname!= null)
      {
          //Create the 3B runner and the control panel, then execute the contro
l panel in the
          //GUI thread.
          final Rectangle roi = ip.getRoi();
          final double pixel_size_in_nm_ = pixel_size_in_nm;
          final ThreeBRunner tbr = new ThreeBRunner(mask, s, cfg, fullname, fir
stfr, lastfr);

          SwingUtilities.invokeLater(
                  new Runnable() {
                          public void run() {
                                  new EControlPanel(roi, pixel_size_in_nm_, fna
me, tbr);
                          }
                  }
          );

      }
  }
```

10.36.3 Member Data Documentation

10.36.3.1 ImagePlus three_B::window [package]

Definition at line 134 of file three_B.java.

Referenced by run(), and setup().

10.36.3.2 ByteProcessor three_B::mask [package]

Definition at line 135 of file three_B.java.

Referenced by run().

10.36.3.3 String three_B::arg [package]

Definition at line 136 of file three_B.java.

Referenced by run(), and setup().

The documentation for this class was generated from the following file:

- three_B.java

10.37 ThreeBDialog Class Reference

Dialog box for starting 3B The dialog highlights bad things in red.

Public Member Functions

- double parseDouble (String s)
- void textValueChanged (TextEvent e)

Package Functions

- ThreeBDialog (int count, int npix_, int nframes_)
- int parseInt (String s)
- TextField getFWHMField ()
- TextField getPixelSizeField ()
- TextField getSpotsField ()
- TextField getFirstFrameField ()
- TextField getLastFrameField ()
- double getFWHM ()
- double getPixelSize ()
- int getSpots ()
- int getFirstFrame ()
- int getLastFrame ()
- int getCount ()

Package Attributes

- int count_
- int npix
- int nframes
- Color c
- Color bg

10.37.1 Detailed Description

Dialog box for starting 3B The dialog highlights bad things in red.

Definition at line 432 of file three_B.java.

10.37.2 Constructor & Destructor Documentation

10.37.2.1 ThreeBDialog::ThreeBDialog (int *count*, int *npix_*, int *nframes_*) [inline, package]

Definition at line 437 of file three_B.java.

References bg, c, count_, getFWHMField(), nframes, npix, and textValueChanged().

```
{
    super("3B Analysis");
    count_ = count;
    npix   = npix_;
    nframes= nframes_;

    addNumericField("Microscope FWHM", 250.0, 1, 10, "nm");
    addNumericField("Pixel size", 100., 1, 10, "nm");
    addNumericField("Initial number of spots", Math.round(count / 10.), 0, 10
  , "spots");
    addNumericField("First frame", 0., 0, 10, "");
    addNumericField("Last frame", nframes-1., 0, 10, "");
    addTextAreas("",null, 8,30);
    getTextArea1().setEditable(false);
    c = getFWHMField().getBackground(); //Get the default background colour
    bg = getBackground(); //Dialog background color
    getTextArea1().removeTextListener(this); //To prevent event thrashing whe
  n we write messages

    //Process initial warnings
    textValueChanged(null);
}
```

10.37.3 Member Function Documentation

10.37.3.1 double ThreeBDialog::parseDouble (String *s*) [inline]

Definition at line 460 of file three_B.java.

Referenced by getFWHM(), and getPixelSize().

```
{
    try
    {
        return Double.parseDouble(s);
    }
    catch(Exception e)
    {
        return 0;
    }
}
```

10.37.3.2 int ThreeBDialog::parseInt (String *s*) [inline, package]

Definition at line 471 of file three_B.java.

Referenced by getFirstFrame(), getLastFrame(), and getSpots().

```
{
    try
    {
        return Integer.parseInt(s);
    }
    catch(Exception e)
    {
        return 0;
    }
}
```

10.37.3.3 TextField ThreeBDialog::getFWHMField ( ) [inline, package]

Definition at line 483 of file three_B.java.

Referenced by getFWHM(), textValueChanged(), and ThreeBDialog().

```
{
    return (TextField)(getNumericFields().get(0));
}
```

10.37.3.4 TextField ThreeBDialog::getPixelSizeField ( ) [inline, package]

Definition at line 487 of file three_B.java.

Referenced by getPixelSize(), and textValueChanged().

```
{
    return (TextField)(getNumericFields().get(1));
}
```

10.37.3.5 TextField ThreeBDialog::getSpotsField ( ) [inline, package]

Definition at line 491 of file three_B.java.

Referenced by getSpots(), and textValueChanged().

```
{
    return (TextField)(getNumericFields().get(2));
}
```

10.37.3.6 TextField ThreeBDialog::getFirstFrameField ( ) [inline, package]

Definition at line 495 of file three_B.java.

Referenced by getFirstFrame(), and textValueChanged().

```
{
    return (TextField)(getNumericFields().get(3));
}
```

10.37.3.7 TextField ThreeBDialog::getLastFrameField ( ) [inline, package]

Definition at line 499 of file three_B.java.

Referenced by getLastFrame(), and textValueChanged().

```
{
    return (TextField)(getNumericFields().get(4));
}
```

10.37.3.8 double ThreeBDialog::getFWHM ( ) [inline, package]

Definition at line 504 of file three_B.java.

References getFWHMField(), and parseDouble().

Referenced by textValueChanged().

```
{
    return parseDouble(getFWHMField().getText());
}
```

10.37.3.9 double ThreeBDialog::getPixelSize ( ) [inline, package]

Definition at line 509 of file three_B.java.

References getPixelSizeField(), and parseDouble().

Referenced by textValueChanged().

```
{
    return parseDouble(getPixelSizeField().getText());
}
```

10.37.3.10 int ThreeBDialog::getSpots ( ) [inline, package]

Definition at line 514 of file three_B.java.

References getSpotsField(), and parseInt().

Referenced by textValueChanged().

```
{
    return parseInt(getSpotsField().getText());
}
```

10.37.3.11 int ThreeBDialog::getFirstFrame ( ) [inline, package]

Definition at line 519 of file three_B.java.

References getFirstFrameField(), and parseInt().

Referenced by textValueChanged().

```
{
    return parseInt(getFirstFrameField().getText());
}
```

10.37.3.12 int ThreeBDialog::getLastFrame ( ) [inline, package]

Definition at line 524 of file three_B.java.

References getLastFrameField(), and parseInt().

Referenced by textValueChanged().

```
{
    return parseInt(getLastFrameField().getText());
}
```

10.37.3.13 int ThreeBDialog::getCount ( ) [inline, package]

Definition at line 529 of file three_B.java.

References count_.

Referenced by textValueChanged().

```
{
    return count_;
}
```

10.37.3.14 void ThreeBDialog::textValueChanged (TextEvent *e*) [inline]

Definition at line 534 of file three_B.java.

References bg, c, getCount(), getFirstFrame(), getFirstFrameField(), getFWHM(), getFWHMField(), getLastFrame(), getLastFrameField(), getPixelSize(), getPixelSizeField(), getSpots(), getSpotsField(), nframes, and npix.

Referenced by ThreeBDialog().

```
{
    boolean long_run=false;
    String err = "";
    //               01234567890123456789012345678901234567890123456789012345678901234567890
    if(getCount() > 1000)
    {
        long_run=true;
        err = err + "Warning: large area selected.\n3B will run very slowly.\
  n";
    }

    if(npix < 2500)
        err = err + "Warning: image is very small. Fitting may be bad because
  \nimage noise cannot be accurately estimated.\n";

    if(getSpots() > 500)
    {
        err = err + "Warning: large number of spots.\n3B will run very slowly
  .\n";
        getSpotsField().setBackground(Color.RED);
    }
    else
        getSpotsField().setBackground(c);


    if(getFWHM() < 200)
    {
        err = err + "Warning: unrealistically small\nmicsoscope resolution.\n
  ";
        getFWHMField().setBackground(Color.RED);
    }
    else if(getFWHM() > 350)
    {
        err = err + "Warning: 3B will not work well with\na poorly focussed m
  icroscope.\n";
        getFWHMField().setBackground(Color.RED);
```

```
  }
  else
      getFWHMField().setBackground(c);


  if(getPixelSize() < 70)
  {
      getPixelSizeField().setBackground(Color.RED);
      err = err + "Warning: Very small pixels specified.\nAre you sure?\n";

  }
  else if(getPixelSize() > 180)
  {
      getPixelSizeField().setBackground(Color.RED);
      err = err + "Warning: 3B will not work well if the camera\nresolution
 is too poor.\n";
  }
  else
      getPixelSizeField().setBackground(c);

  //Clamp the frames
  int first = getFirstFrame();
  int last  = getLastFrame();

  int nfirst = Math.max(0, Math.min(nframes-1, first));
  int nlast = Math.max(nfirst, Math.min(nframes-1, last));

  if(first != nfirst)
      getFirstFrameField().setText(Integer.toString(nfirst));
  if(last != nlast)
      getLastFrameField().setText(Integer.toString(nlast));

  if(last -first + 1 > 500)
  {
      getFirstFrameField().setBackground(Color.RED);
      getLastFrameField().setBackground(Color.RED);
      err = err + "Warning: large number of frames specified.\n3B will run
very slowly and may be inaccurate.\nFewer than 500 frames is strongly recommended
.\n200--300 is generally most suitable.\n";
      long_run = true;
  }
  if(last -first + 1 < 150)
  {
      getFirstFrameField().setBackground(Color.RED);
      getLastFrameField().setBackground(Color.RED);
      err = err + "Warning: small number of frames specified.\n3B may be in
accurate.\nAt least 150 frames is recommended.\n";
  }
  else
  {
      getFirstFrameField().setBackground(c);
      getLastFrameField().setBackground(c);
  }


  if(!long_run && (last -first + 1)*getCount() > 200000)
  {
      err = err + "Warning: large amount of data specified.\n"+
                  "3B will run very slowly.\n"+
                  "Reduce the area and/or number of frames.\n"+
                  "We recommend: \n" +
                  "Number of frames*area in pixels < 200,000.";
```

```
            getFirstFrameField().setBackground(Color.RED);
            getLastFrameField().setBackground(Color.RED);
        }


        if(!err.equals(""))
            getTextArea1().setBackground(Color.RED);
        else
            getTextArea1().setBackground(bg);



        getTextArea1().setText(err);
        repaint();

    }
```

10.37.4 Member Data Documentation

10.37.4.1 int ThreeBDialog::count_ [package]

Definition at line 434 of file three_B.java.

Referenced by getCount(), and ThreeBDialog().

10.37.4.2 int ThreeBDialog::npix [package]

Definition at line 434 of file three_B.java.

Referenced by textValueChanged(), and ThreeBDialog().

10.37.4.3 int ThreeBDialog::nframes [package]

Definition at line 434 of file three_B.java.

Referenced by textValueChanged(), and ThreeBDialog().

10.37.4.4 Color ThreeBDialog::c [package]

Definition at line 435 of file three_B.java.

Referenced by textValueChanged(), and ThreeBDialog().

10.37.4.5 Color ThreeBDialog::bg [package]

Definition at line 435 of file three_B.java.

Referenced by textValueChanged(), and ThreeBDialog().

The documentation for this class was generated from the following file:

- three_B.java

10.38 ThreeBGlobalConstants Class Reference

Static Public Attributes

- static int critical_iterations = 200

10.38.1 Detailed Description

Definition at line 29 of file three_B.java.

10.38.2 Member Data Documentation

10.38.2.1 int ThreeBGlobalConstants::critical_iterations = 200 [static]

Definition at line 31 of file three_B.java.

Referenced by ThreeBRunner::send_new_points(), and EControlPanel::update_canvas().

The documentation for this class was generated from the following file:

- three_B.java

10.39 ThreeBHelp Class Reference

3B plugin cclass to bring up a basic help window.

Public Member Functions

- void run (String arg)

10.39.1 Detailed Description

3B plugin cclass to bring up a basic help window.

Definition at line 17 of file ThreeBHelp.java.

10.39.2 Member Function Documentation

10.39.2.1 void ThreeBHelp::run (String *arg*) [inline]

Definition at line 19 of file ThreeBHelp.java.

References Util::read().

```
{
    String path = getClass().getResource("img/").toString();
    Reader htmlstream = new InputStreamReader(getClass().getClassLoader().get
  ResourceAsStream("instructions.html"));
    String html = Util.read(htmlstream);

    html = html.replaceAll("img/", path);

    JFrame w = new JFrame("3B help");

    JEditorPane l = new JEditorPane();
    try{
        l.setPage(getClass().getResource("instructions.html").toString());
    }
    catch(IOException e)
    {}
    //l.resize(new java.awt.Dimension(800,600));

    JScrollPane s = new JScrollPane(l, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

    java.awt.Dimension  d = new java.awt.Dimension(600, 600);

    w.add(s);
    w.pack();
    w.setSize(d);
    w.setVisible(true);

}
```

The documentation for this class was generated from the following file:

- ThreeBHelp.java

10.40 ThreeBLoader Class Reference

Plugin class to load up an old 3B run.

Public Member Functions

- void run (String arg)

Private Member Functions

- void close (InputStream i)

10.40.1 Detailed Description

Plugin class to load up an old 3B run.

Definition at line 27 of file ThreeBLoader.java.

10.40.2 Member Function Documentation

10.40.2.1 void ThreeBLoader::close (InputStream *i*) [inline, private]

Definition at line 29 of file ThreeBLoader.java.

Referenced by run().

```
{
    //Not sure what to do here...
    if(i != null)
    {
        try{
            i.close();
        }
        catch(IOException ioe)
        {
            Toolkit.getDefaultToolkit().beep();
            ij.IJ.showStatus("Error closing file: " + ioe.getMessage());
        }
    }
}
```

10.40.2.2 void ThreeBLoader::run (String *arg*) [inline]

Definition at line 45 of file ThreeBLoader.java.

References EControlPanel::append(), close(), EControlPanel::send_status_text_message(), EControlPanel::send_update_canvas_event(), Spot::x, and Spot::y.

```
                            {
        InputStream in = null;
        String name;
        double ps;

        try{
            if(arg.equals("test"))
            {
                name = "test_data.txt";

                try{
```

```
            in = new GZIPInputStream(getClass().getClassLoader().getResou
rceAsStream("test_data.txt.gz"));
        }
        catch(IOException ioe)
        {
            Toolkit.getDefaultToolkit().beep();
            ij.IJ.showStatus("Could not open test data: " + ioe.getMessag
e());
            return;
        }
        ps=100;
    }
    else
    {
        OpenDialog o = new OpenDialog("Open 3B run...", null);
        name = o.getFileName();

        try{
            in = new FileInputStream(o.getDirectory() + o.getFileName());

        }
        catch(java.io.FileNotFoundException ferr)
        {
            Toolkit.getDefaultToolkit().beep();
            ij.IJ.showStatus("Error opening file: " + ferr.getMessage());

            return;
        }



        GenericDialog g = new GenericDialog("Pixel size");
        g.addNumericField("Pixel size", 100., 0, 5, "nm");
        g.showDialog();
        ps = g.getNextNumber();
    }

    //Yay @ cargoculting
    InputStreamReader d = new InputStreamReader(in);
    BufferedReader r = new BufferedReader(d);

    String line;
    final ArrayList<Spot> spots = new ArrayList<Spot>();
    int iterations=0;
    Rectangle roi=null;
    final double pixel_size_in_nm_ = ps;

    while((line = r.readLine()) != null)
    {
        String tokens[] = line.split("\\p{Space}+");

        if(tokens[0].equals("PIXELS"))
        {
            if((tokens.length - 1)%2 != 0)
                throw new IOException("corrupt file (bad pixels line)");

            for(int i=1; i < tokens.length; i+= 2)
            {
                int x, y;
                try
                {
                    x = Integer.parseInt(tokens[i+0]);
```

```
                        y = Integer.parseInt(tokens[i+1]);
                    }
                    catch(NumberFormatException nerr)
                    {
                        throw new IOException("corrupt file (bad pixel coordi
nate)");
                    }

                    if(roi == null)
                        roi = new Rectangle(x, y, 1, 1);
                    else
                        roi.add(x, y);
                }
            }

            if(tokens[0].matches("PASS[0-9]+:"))
            {
                iterations++;

                if((tokens.length - 1)%4 != 0)
                    throw new IOException("corrupt file (pad pass line)");

                for(int i=1; i < tokens.length; i+= 4)
                {
                    Spot s = new Spot();
                    try
                    {
                        s.x = Double.parseDouble(tokens[i+2]);
                        s.y = Double.parseDouble(tokens[i+3]);
                    }
                    catch(NumberFormatException nerr)
                    {
                        throw new IOException("corrupt file (bad spot positio
n)");
                    }

                    spots.add(s);
                }
            }

        }
        if(roi == null)
            throw new IOException("corrupt file (no ROI)");

        final String fname = name;
        final Rectangle roi_ = roi;
        final int its = iterations;
        SwingUtilities.invokeLater(
        new Runnable() {
                public void run() {
                        EControlPanel e = new EControlPanel(roi_, pixel_size_
in_nm_, fname, null);
                        e.append(spots, its);
                        e.send_update_canvas_event();
                        e.send_status_text_message("Using " + fname + ": " +
Integer.toString(its) + " iterations.");
                }
        }
        );
```

```
        }
        catch(IOException another_ioe){
            Toolkit.getDefaultToolkit().beep();
            ij.IJ.showStatus("Error reading data: " + another_ioe.getMessage());
        }
        finally{
            close(in);
        }

    }
```

The documentation for this class was generated from the following file:

- ThreeBLoader.java

10.41 ThreeBRunner Class Reference

This class deals with running the actual 3B code It should be run in a separate thread It will make calls back to a viewer, and accepts termination calls as well.

Public Member Functions

- void register (EControlPanel c)
- void run ()

Static Public Member Functions

- static void addLibraryPath (String pathToAdd) throws Exception

Package Functions

- ThreeBRunner (ByteProcessor mask_, ImageStack s, String cfg_, String filename_-, int firstfr, int lastfr)
- void stop_thread ()
- boolean has_stopped ()
- void send_message_string (String s)
- void send_new_points (float[] points)
- void die (String err)
- boolean should_stop ()
- native void call (String cfg, float[][] images, byte[] mask, int n_images, int rows, int cols, String file)

Static Package Functions

- static String get_plugin_dir ()
- static UnsatisfiedLinkError try_to_load_dlls (String[] s)
- [static initializer]

Package Attributes

- ByteProcessor mask
- float[][] pixels
- EControlPanel cp
- String config
- String filename
- long start_time
- int its

Static Private Member Functions

- static String decodePercent (String str)

Private Attributes

- volatile boolean stop = false
- volatile boolean stopped = false
- boolean fatal = false

10.41.1 Detailed Description

This class deals with running the actual 3B code It should be run in a separate thread It will make calls back to a viewer, and accepts termination calls as well.

Definition at line 1283 of file three_B.java.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 ThreeBRunner::ThreeBRunner (ByteProcessor *mask_*, ImageStack *s*, String *cfg_*, String *filename_*, int *firstfr*, int *lastfr*) [inline, package]

Definition at line 1296 of file three_B.java.

References config, filename, mask, and pixels.

```
{
    config = cfg_;
    filename = filename_;
```

```
        //Take a copy of the mask to stop it getting trashed by other threads
        mask = (ByteProcessor)mask_.duplicate();

        //Convert all the input images into float as a common
        //format. They will be scaled differently from float images
        //loaded by libcvd, but the later normalization will take care of that
        //
        //Oh, and ImageStack counts from 1, not 0
        pixels = new float[lastfr - firstfr + 1][];
        for(int i=firstfr; i <= lastfr; i++)
            pixels[i-firstfr] = (float[])s.getProcessor(i+1).convertToFloat().get
      Pixels();
    }
```

10.41.3 Member Function Documentation

10.41.3.1 void ThreeBRunner::register (EControlPanel *c*) [inline]

Definition at line 1314 of file three_B.java.

References cp.

Referenced by EControlPanel::EControlPanel().

```
    {
        cp = c;
    }
```

10.41.3.2 void ThreeBRunner::stop_thread ( ) [inline, package]

Definition at line 1319 of file three_B.java.

References send_message_string(), and stop.

Referenced by EControlPanel::issue_stop(), and EControlPanel::windowClosing().

```
    {
        stop=true;
        send_message_string("stopping...");
    }
```

10.41.3.3 boolean ThreeBRunner::has_stopped ( ) [inline, package]

Definition at line 1325 of file three_B.java.

References stopped.

Referenced by EControlPanel::windowClosing().

```
{
    return stopped;
}
```

10.41.3.4 void ThreeBRunner::send_message_string (String *s*) [inline, package]

Definition at line 1332 of file three_B.java.

References cp, and EControlPanel::send_status_text_message().

Referenced by stop_thread().

```
{
    cp.send_status_text_message(s);
}
```

10.41.3.5 void ThreeBRunner::send_new_points (float[] *points*) [inline, package]

Definition at line 1337 of file three_B.java.

References EControlPanel::append(), cp, ThreeBGlobalConstants::critical_iterations, its, EControlPanel::send_time_text_message(), EControlPanel::send_update_canvas_event(), start_time, Spot::x, and Spot::y.

```
{
    //New points are sent in the form x, y, x, y
    ArrayList<Spot> tmp = new ArrayList<Spot>();
    for(int i=0; i < points.length; i+=2)
    {
        Spot s = new Spot();
        s.x = points[i];
        s.y = points[i+1];
        tmp.add(s);
    }

    cp.append(tmp, its);
    cp.send_update_canvas_event();

    //This happens each "iteration", i.e. each pass.
    long current = (new java.util.Date()).getTime();

    if(its > 0)
    {
        if(its < 8)
            cp.send_time_text_message("computing...");
        else
        {
            long time_per_it = (current -start_time) / its;
            int target = (int)Math.ceil(its * 1.0 / ThreeBGlobalConstants.
  critical_iterations) * ThreeBGlobalConstants.critical_iterations;
            int its_remaining = target-its;

            System.out.println("Its = " + its);
            System.out.println("rem = " + its_remaining);
```

```
                System.out.println("time_per_it = " + time_per_it);


                long time_remaining_s = (its_remaining * time_per_it)/1000;

                long s = time_remaining_s % 60;
                long m = (time_remaining_s / 60)%60;
                long h = time_remaining_s / 3600;

                cp.send_time_text_message( h + "h" + m + "m (for " + target + " i
    terations)");
            }

        }

        //Increment after, since it outputs the initial spots as well
        its++;
    }
```

10.41.3.6 void ThreeBRunner::die (String *err*) [inline, package]

Definition at line 1385 of file three_B.java.

References cp, EControlPanel::die(), and fatal.

```
    {
        cp.die(err);
        fatal=true;
    }
```

10.41.3.7 boolean ThreeBRunner::should_stop ( ) [inline, package]

Definition at line 1391 of file three_B.java.

References stop.

```
    {
        return stop;
    }
```

10.41.3.8 native void ThreeBRunner::call (String *cfg,* float *images[][],* byte[] *mask,* int *n_images,* int *rows,* int *cols,* String *file*) [package]

Referenced by run().

10.41.3.9 void ThreeBRunner::run ( ) [inline]

Definition at line 1398 of file three_B.java.

References call(), config, cp, fatal, filename, its, mask, pixels, EControlPanel::send_-status_text_message(), start_time, and stopped.

```
{
    System.out.println("About to call...");
    start_time = (new java.util.Date()).getTime();
    its=0;

    call(config, pixels, (byte[])mask.getPixels(), pixels.length, mask.getHei
  ght(), mask.getWidth(), filename);

    System.out.println("Finished.");

    if(!fatal)
    {
        cp.send_status_text_message("Finished\n");
        ij.IJ.showStatus("3B run terminated");
    }

    stopped=true;
}
```

10.41.3.10 static String ThreeBRunner::decodePercent (String *str*) [inline, static, private]

Definition at line 1420 of file three_B.java.

```
{
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < str.length(); i++)
    {
        char c = str.charAt(i);
        if(c == '+')
            sb.append(' ');
        else if(c == '%')
        {
            sb.append((char) Integer.parseInt(str.substring(i + 1, i + 3), 16
  ));
            i += 2;
        }
        else
            sb.append(c);
    }
    return sb.toString();
}
```

10.41.3.11 static void ThreeBRunner::addLibraryPath (String *pathToAdd*) throws Exception [inline, static]

Adds the specified path to the java library path.

Parameters

pathToAdd	the path to add

Exceptions

Exception	

Definition at line 1445 of file three_B.java.

```
                                                                    {
    final Field usrPathsField = ClassLoader.class.getDeclaredField("usr_paths
  ");
    usrPathsField.setAccessible(true);

    //get array of paths
    final String[] paths = (String[])usrPathsField.get(null);

    //check if the path to add is already present
    for(String path : paths) {
        if(path.equals(pathToAdd)) {
            return;
        }
    }

    //add the new path
    final String[] newPaths = Arrays.copyOf(paths, paths.length + 1);
    newPaths[newPaths.length-1] = pathToAdd;
    usrPathsField.set(null, newPaths);
}
```

10.41.3.12 static String ThreeBRunner::get_plugin_dir ( ) [inline, static, package]

Definition at line 1489 of file three_B.java.

Referenced by try_to_load_dlls().

```
{
    try{
        String img_url = (new SPair()).getClass().getResource("img").getFile(
  );
        URI jar_url = null;

        jar_url= new URI(img_url.substring(0, img_url.length()-5));

        File jar = new File(jar_url);

        System.out.println("File: " + jar.getCanonicalPath());


        String dir = jar.getParentFile().getCanonicalPath() + File.separator;

        System.out.println("Dir: " + dir);

        return dir;

    }
    catch(Exception e){
        return "";
    }
}
```

10.41.3.13 static UnsatisfiedLinkError ThreeBRunner::try_to_load_dlls (String[] *s*) [inline, static, package]

Definition at line 1514 of file three_B.java.

References get_plugin_dir().

```
{
    String dir = get_plugin_dir();

    try{
        for(int i=0; i < s.length; i++)
        {
            System.out.println("Loading " + dir+s[i]);
            System.load(dir + s[i]);
        }

        return null;
    }
    catch(UnsatisfiedLinkError e)
    {
        System.out.println("Link error: " + e.getMessage());
        return e;
    }

}
```

10.41.3.14 ThreeBRunner::[static initializer] ( ) [inline, static, package]

10.41.4 Member Data Documentation

10.41.4.1 ByteProcessor ThreeBRunner::mask [package]

Definition at line 1285 of file three_B.java.

Referenced by run(), and ThreeBRunner().

10.41.4.2 float [][] ThreeBRunner::pixels [package]

Definition at line 1286 of file three_B.java.

Referenced by run(), and ThreeBRunner().

10.41.4.3 volatile boolean ThreeBRunner::stop = false [private]

Definition at line 1287 of file three_B.java.

Referenced by should_stop(), and stop_thread().

10.41.4.4 volatile boolean ThreeBRunner::stopped = false [private]

Definition at line 1288 of file three_B.java.

Referenced by has_stopped(), and run().

10.41.4.5 boolean ThreeBRunner::fatal = false [private]

Definition at line 1289 of file three_B.java.

Referenced by die(), and run().

10.41.4.6 EControlPanel ThreeBRunner::cp [package]

Definition at line 1290 of file three_B.java.

Referenced by die(), register(), run(), send_message_string(), and send_new_points().

10.41.4.7 String ThreeBRunner::config [package]

Definition at line 1291 of file three_B.java.

Referenced by run(), and ThreeBRunner().

10.41.4.8 String ThreeBRunner::filename [package]

Definition at line 1292 of file three_B.java.

Referenced by run(), and ThreeBRunner().

10.41.4.9 long ThreeBRunner::start_time [package]

Definition at line 1293 of file three_B.java.

Referenced by run(), and send_new_points().

10.41.4.10 int ThreeBRunner::its [package]

Definition at line 1294 of file three_B.java.

Referenced by run(), and send_new_points().

The documentation for this class was generated from the following file:

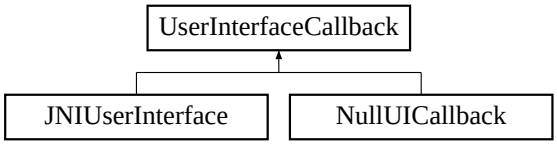
- three_B.java

10.42 UserInterfaceCallback Class Reference

Callback class used by FitSpots to provide enough hooks for a user interface.

`#include <multispot5.h>`

Inheritance diagram for UserInterfaceCallback:



Classes

- struct UserIssuedStop

Public Member Functions

- virtual void per_spot (int iteration, int pass, int spot_num, int total_spots)=0
- virtual void per_modification (int iteration, int spot_num, int total_spots)=0
- virtual void per_pass (int iteration, int pass, const std::vector< TooN::Vector< 4 > > &spots)=0
- virtual void perhaps_stop ()=0
- virtual ~UserInterfaceCallback ()

10.42.1 Detailed Description

Callback class used by FitSpots to provide enough hooks for a user interface.

Definition at line 63 of file multispot5.h.

10.42.2 Constructor & Destructor Documentation

10.42.2.1 UserInterfaceCallback::~UserInterfaceCallback ( ) [virtual]

Empty destructor.

Definition at line 49 of file multispot5.cc.

```
{}
```

10.42.3 Member Function Documentation

10.42.3.1 virtual void UserInterfaceCallback::per_spot (int *iteration*, int *pass*, int *spot_num*, int *total_spots*) [pure virtual]

This function is called once per spot in each pass.

The idea is to provide a display along the lines of: Iteration #1 optimizing #2% complete

Parameters

iteration	Iteration number
pass	Pass number
spot_num	Spot currently being optimized
total_spots	Total number of spots to be optimized

Implemented in NullUICallback, and JNIUserInterface.

10.42.3.2 virtual void UserInterfaceCallback::per_modification (int *iteration*, int *spot_num*, int *total_spots*) [pure virtual]

This function is called once per spot in the modification phase.

The idea is to provide a display along the lines of: Iteration #1 modifying #2% complete

Parameters

iteration	Iteration number
spot_num	Spot currently being optimized
total_spots	Total number of spots to be optimized

Implemented in NullUICallback, and JNIUserInterface.

10.42.3.3 virtual void UserInterfaceCallback::per_pass (int *iteration*, int *pass*, const std::vector< TooN::Vector< 4 > > & *spots*) [pure virtual]

This function is called once each time PASS data is outputted.

It will allow the GUI to build up a reconstruction.

Parameters

iteration	Iteration number
pass	Pass number
spots	Data to be reconstructed

Implemented in NullUICallback, and JNIUserInterface.

10.42.3.4 virtual void UserInterfaceCallback::perhaps_stop ( ) [pure virtual]

The user wishes to issue a stop instruction to the program (perhaps done via an asynchronus call to an instance of of UserInterfaceCallback).

This function is called as often as possible and will throw UserIssuedStop when the condition is met.

Implemented in NullUICallback, and JNIUserInterface.

The documentation for this class was generated from the following files:

- multispot5.h
- multispot5.cc

10.43 UserInterfaceCallback::UserIssuedStop Struct Reference

`#include <multispot5.h>`

10.43.1 Detailed Description

Definition at line 95 of file multispot5.h.

The documentation for this struct was generated from the following file:

- multispot5.h

10.44 Util Class Reference

Utility calss to hold a number of handy static functions.

Static Package Functions

- static String read (Reader in)
- static SPair getFileName (String t)

10.44.1 Detailed Description

Utility calss to hold a number of handy static functions.

Definition at line 45 of file three_B.java.

10.44.2 Member Function Documentation

10.44.2.1 static String Util::read (Reader *in*) [inline, static, package]

Read a file into a string.

It simply sidcards errors because if the file is missing from the JAR archive, then extreme badness has happened and I have no idea how to recover.

Parameters

in	File to be read

Returns

file contents in a string

Definition at line 53 of file three_B.java.

Referenced by ThreeBHelp::run(), and three_B::run().

```
{
    try {
        final char[] buffer = new char[100];
        StringBuilder out = new StringBuilder();
        int read;
        do
        {
            read = in.read(buffer, 0, buffer.length);
            if (read>0)
                out.append(buffer, 0, read);
        } while (read>=0);
        return out.toString();
    }
    catch(java.io.IOException err)
    {
        return "";
        //What do we do here?
    }
}
```

The documentation for this class was generated from the following file:

- three_B.java

11 File Documentation

11.1 ClassicGlow.java File Reference

Classes

- class ClassicGlow

  Plugin implementing the classic glow/hot LUT which seems to be missing from ImageJ.

11.2 conjugate_gradient_only.h File Reference

```
#include <TooN/TooN.h>
#include <utility>
#include <cstdlib>
```

Classes

- struct ConjugateGradientOnly< Size, Precision >

  Class for performing optimization with Conjugate Gradient, where only the derivatives are available.

11.3 debug.cc File Reference

Debugging bits.

Functions

- Image< byte > scale_to_bytes (const Image< float > &im, float lo, float hi)
- void test_output_patch_variance (const vector< Image< float > > &ims)

11.3.1 Detailed Description

Debugging bits.

Definition in file debug.cc.

11.4 debug.h File Reference

Debugging bits.

```
#include <cassert>
```

Functions

- template<class C >
  void assert_same_size (const C &images)

11.4.1 Detailed Description

Debugging bits.

Definition in file debug.h.

11.5 documentation.h File Reference

Doxygen documentation bits.

11.5.1 Detailed Description

Doxygen documentation bits.

Definition in file documentation.h.

11.6 forward_algorithm.h File Reference

Contains an implementation fo the forward algorithm.

```
#include <tr1/tuple>
#include <tr1/array>
#include <TooN/TooN.h>
#include <vector>
#include <cmath>
```

Functions

- double ln (double x)
- template<int States, class Btype , class Otype >
  std::tr1::tuple< double, TooN::Vector< Btype::NumParameters >, TooN::Matrix< Btype::NumParameters > > forward_algorithm_hessian (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O, bool compute_deriv=1, bool compute_hessian=1)
- template<int States, class Btype , class Otype >
  double forward_algorithm (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)

- template<int States, class Btype , class Otype >
  std::pair< double, TooN::Vector< Btype::NumParameters > > forward_algorithm_deriv (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)
- template<int States, class Btype , class Otype >
  std::vector< std::tr1::array< double, States > > forward_algorithm_delta (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)
- template<int States, class Btype , class Otype >
  void forward_algorithm_delta2 (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O, std::vector< std::tr1::array< double, States > > &delta)
- template<int States, class Btype , class Otype >
  std::pair< std::vector< std::tr1::array< double, States > >, std::vector< std::tr1::array< double, States > > > forward_backward_algorithm (TooN::Matrix< States > A, TooN::Vector< States > pi, const Btype &B, const std::vector< Otype > &O)
- template<class A , class Rng >
  int select_random_element (const A &v, const double scale, Rng &rng)
- template<int N, class Rng >
  int sample_unscaled_log (std::tr1::array< double, N > a, Rng &rng)
- template<int States, class StateType , class Rng >
  std::vector< StateType > backward_sampling (TooN::Matrix< States > A, const std::vector< std::tr1::array< double, States > > &delta, Rng &rng)

11.6.1 Detailed Description

Contains an implementation fo the forward algorithm.

Definition in file forward_algorithm.h.

11.7 LoadTestData.java File Reference

Classes

- class LoadTestData

  Plugin class to load the standard test data from the JAR file.

11.8 mersenne.cpp File Reference

Agner Fogg's Mersenne Twister implementation.

```
#include "randomc.h"
```

11.8.1 Detailed Description

Agner Fogg's Mersenne Twister implementation.

Definition in file mersenne.cpp.

11.9 mt19937.h File Reference

Mersenne twister interface code.

```
#include "randomc.h"
#include <iostream>
#include <sstream>
#include <string>
#include <cmath>
#include <iomanip>
```

Classes

- struct MT19937

  Useful wrapper for MT19937 random number generator class.
- struct MT19937::ParseError

  Null struct thrown if attempting to load state from stream yields a parse error.

11.9.1 Detailed Description

Mersenne twister interface code.

Definition in file mt19937.h.

11.10 multispot5.cc File Reference

Fit spots to the data.

```
#include <cstdlib>
#include <cerrno>
#include <cstring>
#include <stack>
#include <algorithm>
#include <climits>
```

```
#include <iomanip>
#include <map>
#include <tr1/memory>
#include <cvd/image_io.h>
#include <cvd/image_convert.h>
#include <cvd/morphology.h>
#include <cvd/connected_components.h>
#include <cvd/draw.h>
#include <cvd/vector_image_ref.h>
#include <cvd/random.h>
#include <cvd/timer.h>
#include <gvars3/instances.h>
#include <TooN/functions/derivatives.h>
#include <TooN/determinant.h>
#include <TooN/SymEigen.h>
#include <TooN/optimization/conjugate_gradient.h>
#include "conjugate_gradient_only.h"
#include "forward_algorithm.h"
#include "numerical_derivatives.h"
#include "storm.h"
#include "storm_imagery.h"
#include "debug.h"
#include "sampled_multispot.h"
#include "mt19937.h"
#include "utility.h"
#include "multispot5.h"
```

Classes

- class NullUICallback

  User interface callback class which does nothing.
- class NullGraphics

  Graphics class which does absoloutely nothing.
- class DataForMCMC

  Closure hoding the data required do use GibbsSampler2 See FitSpots for naming of variables.

- class Kahan

  Class implementing the Kahan summation algorithm to allow accurate summation of very large numbers of doubles.
- class NegativeFreeEnergy

  Class for computing the negitve free energy using thermodynamic integration.
- struct IndexLexicographicPosition< Cmp, First >

  Class for sorting a list of indexes to an array of spots lexicographically according to the 2D positions of the spots.
- struct SampledBackgroundData

  Closure holding image data generated using samples drawn from the model.
- struct SpotNegProbabilityDiffWithSampledBackground

  Compute the derivative of the negative log probability with respect to the parameters of one spot, given some samples of the other spots.
- class FreeEnergyHessian

  Class for computing the Hessian of the negative free energy.
- struct LessSecond

  Comparator functor for the first element of a std::pair.
- class FitSpots

  Mega class which actually does the meat of the spot fitting.

Defines

- #define TIME(X)

Functions

- auto_ptr< UserInterfaceCallback > null_ui ()
- auto_ptr< FitSpotsGraphics > null_graphics ()
- Vector spots_to_Vector (const vector< Vector< 4 > > &s)
- vector< Vector< 4 > > spots_to_vector (const Vector<> &s)
- Image< byte > scale_to_bytes (const Image< float > &im, float lo, float hi)
- Image< byte > scale_to_bytes (const Image< float > &im)
- Image< float > average_image (const vector< Image< float > > &ims)
- Matrix< 4 > sampled_background_spot_hessian_ffbs (const Vector< 4 > &spot, const SampledBackgroundData &d, int bs_iterations, MT19937 &rng)
- Matrix< 4 > sampled_background_spot_hessian2 (const Vector< 4 > &spot, const SampledBackgroundData &d)
- Matrix< 4 > sampled_background_spot_hessian_FAKE (const Vector< 4 > &spot, const SampledBackgroundData &d)
- void get_spot_pixels (const vector< ImageRef > &pixels, const Vector< 4 > &spot, vector< int > &out)
- vector< string > split (const string &line)
- template<class C >
  string xtoa (const C &x)

- template<class C >
  C atox (const string &s, const string &msg)
- StateParameters parse_log_file (istream &in)
- StateParameters generate_state_parameters_ye_olde (const BasicImage< double > &log_ratios, const vector< Image< float > > &ims, vector< ImageRef > pixels)
- set< ImageRef > dilate_mask (const vector< ImageRef > &v, double r)
- double brightness_motion_limit (double mu, double sigma, bool not_one)
- void fit_spots_new (const vector< Image< float > > &ims, StateParameters &p, ofstream &save_spots, FitSpotsGraphics &gr)
- void fit_spots_new (const vector< Image< float > > &ims, StateParameters &p, ofstream &save_spots, FitSpotsGraphics &gr, UserInterfaceCallback &ui)

11.10.1 Detailed Description

Fit spots to the data.

Definition in file multispot5.cc.

11.10.2 Define Documentation

11.10.2.1 #define TIME(*X*)

Definition at line 39 of file multispot5.cc.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes().

11.10.3 Function Documentation

11.10.3.1 Matrix<4> sampled_background_spot_hessian_ffbs (const Vector< 4 > & *spot*, const SampledBackgroundData & *d*, int *bs_iterations*, MT19937 & *rng*)

Compute the Hessian of the log probability.

The background is sampled rather sparsely, and the spot in question is sampled much more densely using FFBS.

Parameters

spot	Spot parameters
d	Background brightness (from other spots)
bs_iterations	Exter backward sampling iterations
rng	Random number generator

Returns

the Hessian of the log probability around the spot

Definition at line 742 of file multispot5.cc.

References SampledBackgroundData::A, SampledMultispot::compute_spot_intensity(), SampledMultispot::compute_spot_intensity_hessian(), diff_log_log_normal(), forward_-algorithm_delta(), hess_log_log_normal(), SampledBackgroundData::mu_blur, SampledBackgroundData::mu_-brightness, SampledBackgroundData::O, SampledBackgroundData::pi, SampledBackgroundData::pixel_-intensities, SampledBackgroundData::pixels, SampledBackgroundData::sample_intensities_-without_spot, SampledBackgroundData::sigma_blur, SampledBackgroundData::sigma_-brightness, and SampledBackgroundData::variance.

Referenced by FitSpots::optimize_each_spot_in_turn_for_several_passes(), and FitSpots::try_-modifying_model().

```
{
    vector<tuple<double, Vector<4>, Matrix<4> > > spot_hess_etc =
      compute_spot_intensity_hessian(d.pixels, spot);
    vector<double> spot_intensities = compute_spot_intensity(d.pixels, spot);

    Matrix<4> sum_hess_log = Zeros;
    Matrix<4> sum_diff2_log = Zeros;

    vector<State> current_sample;

    const unsigned int nframes = d.pixel_intensities.size();
    const unsigned int npixels = d.pixels.size();

    Matrix<4> sum_hess  = Zeros;
    Vector<4> sum_deriv = Zeros;

    vector<pair<Matrix<4>, Vector<4> > > hess_and_deriv_part(nframes);

    for(unsigned int s=0; s < d.sample_intensities_without_spot.size(); s++)
    {
        SpotWithBackground B(d.sample_intensities_without_spot[s], spot_intensiti
      es, d.pixel_intensities, d.variance);

        //Compute what the per-frame hess and deriv parts are
        //if the spot is on in a frame.
        for(unsigned int frame=0; frame < nframes; frame++)
        {
            Matrix<4> hess = Zeros;
            Vector<4> deriv = Zeros;

            for(unsigned int pixel=0; pixel < npixels; pixel++)
            {
                double e = d.pixel_intensities[frame][pixel] - (d.
      sample_intensities_without_spot[s][frame][pixel] + spot_intensities[pixel]);
                //Build up the derivative
                hess += e * get<2>(spot_hess_etc[pixel]) - get<1>(spot_hess_etc[p
      ixel]).as_col() * get<1>(spot_hess_etc[pixel]).as_row();
                deriv += e * get<1>(spot_hess_etc[pixel]);
            }
            hess_and_deriv_part[frame] = make_pair(hess, deriv);
        }

        //Forward filtering
```

```
        std::vector<array<double, 3> > delta = forward_algorithm_delta(d.A, d.pi,
       B, d.O);

        for(int i=0; i < bs_iterations; i++)
        {
            current_sample = backward_sampling<3,State>(d.A, delta, rng);

            Matrix<4> hess = Zeros;
            Vector<4> deriv = Zeros;
            for(unsigned int frame=0; frame < nframes; frame++)
                if(current_sample[frame] == 0)
                {
                        hess += hess_and_deriv_part[frame].first;
                        deriv += hess_and_deriv_part[frame].second;
                }

            sum_hess += hess + deriv.as_col() * deriv.as_row();
            sum_deriv += deriv;
        }
    }

    sum_hess  /= (bs_iterations * d.sample_intensities_without_spot.size() * d.
      variance);
    sum_deriv /= (bs_iterations * d.sample_intensities_without_spot.size() * d.
      variance);

    sum_hess -= sum_deriv.as_col() * sum_deriv.as_row();

    sum_hess[0][0] += hess_log_log_normal(spot[0], d.mu_brightness, d.
      sigma_brightness);
    sum_hess[1][1] += hess_log_log_normal(spot[1], d.mu_blur, d.sigma_blur);
    sum_deriv[0] += diff_log_log_normal(spot[0], d.mu_brightness, d.
      sigma_brightness);
    sum_deriv[1] += diff_log_log_normal(spot[1], d.mu_blur, d.sigma_blur);

    //cout << "Turboderiv:" << sum_deriv << endl;
    //cout << "Turbohess:\n" << sum_hess << endl;

    return sum_hess;
}
```

11.10.3.2 Matrix<4> sampled_background_spot_hessian2 (const Vector< 4 > & *spot,* const SampledBackgroundData & *d*)

Debugging function. Not mathematically correct. Do not use.

Definition at line 819 of file multispot5.cc.

References SampledBackgroundData::A, SampledMultispot::compute_spot_intensity_-hessian(), diff_log_log_normal(), forward_algorithm_hessian(), hess_log_log_normal(), SampledBackgroundData::mu_blur, SampledBackgroundData::mu_brightness, Sampled-BackgroundData::O, SampledBackgroundData::pi, SampledBackgroundData::pixel_intensities, SampledBackgroundData::pixels, SampledBackgroundData::sample_intensities_without_-spot, SampledBackgroundData::sigma_blur, SampledBackgroundData::sigma_brightness, and SampledBackgroundData::variance.

```
{
```

```
    vector<tuple<double, Vector<4>, Matrix<4> > > spot_intensities =
      compute_spot_intensity_hessian(d.pixels, spot);

    Matrix<4> sum_hess_log = Zeros;
    Matrix<4> sum_diff2_log = Zeros;

    for(unsigned int s=0; s < d.sample_intensities_without_spot.size(); s++)
    {
        SpotWithBackground B(d.sample_intensities_without_spot[s], spot_intensiti
      es, d.pixel_intensities, d.variance);

        double prob;
        Vector<4> diff;
        Matrix<4> hess;

        tie(prob, diff, hess) = forward_algorithm_hessian(d.A, d.pi, B, d.O);

        sum_hess_log += hess;

        diff += makeVector(diff_log_log_normal(spot[0], d.mu_brightness, d.
      sigma_brightness), diff_log_log_normal(spot[1], d.mu_blur, d.sigma_blur), 0, 0);
        sum_diff2_log += diff.as_col() * diff.as_row();
    }

    Matrix<4> hess_log = sum_hess_log / d.sample_intensities_without_spot.size();

    Matrix<4> diff2_log = sum_diff2_log / d.sample_intensities_without_spot.size(
      );

    //Add in the prior

    hess_log[0][0] += hess_log_log_normal(spot[0], d.mu_brightness, d.
      sigma_brightness);
    hess_log[1][1] += hess_log_log_normal(spot[1], d.mu_blur, d.sigma_blur);

    return hess_log + diff2_log;
}
```

11.10.3.3 Matrix<4> sampled_background_spot_hessian_FAKE (const Vector< 4 > & *spot*, const SampledBackgroundData & *d*)

Debugging function. Not mathematically correct. Do not use.

Definition at line 854 of file multispot5.cc.

References SampledBackgroundData::A, SampledMultispot::compute_spot_intensity_-hessian(), forward_algorithm_hessian(), hess_log_log_normal(), SampledBackgroundData::mu_-blur, SampledBackgroundData::mu_brightness, SampledBackgroundData::O, Sampled-BackgroundData::pi, SampledBackgroundData::pixel_intensities, SampledBackground-Data::pixels, SampledBackgroundData::sample_intensities_without_spot, SampledBackgroundData::sigma_-blur, SampledBackgroundData::sigma_brightness, and SampledBackgroundData::variance.

```
{
    vector<tuple<double, Vector<4>, Matrix<4> > > spot_intensities =
      compute_spot_intensity_hessian(d.pixels, spot);

    Matrix<4> sum_hess_log = Zeros;
```

```
    for(unsigned int s=0; s < d.sample_intensities_without_spot.size(); s++)
    {
        SpotWithBackground B(d.sample_intensities_without_spot[s], spot_intensiti
      es, d.pixel_intensities, d.variance);

        double prob;
        Vector<4> diff;
        Matrix<4> hess;

        tie(prob, diff, hess) = forward_algorithm_hessian(d.A, d.pi, B, d.O);

        sum_hess_log += hess;
    }

    Matrix<4> hess_log = sum_hess_log / d.sample_intensities_without_spot.size();


    //Add in the prior

    hess_log[0][0] += hess_log_log_normal(spot[0], d.mu_brightness, d.
      sigma_brightness);
    hess_log[1][1] += hess_log_log_normal(spot[1], d.mu_blur, d.sigma_blur);

    return hess_log;
}
```

11.10.3.4 StateParameters generate_state_parameters_ye_olde (const BasicImage< double > & *log_ratios,* const vector< Image< float > > & *ims,* vector< ImageRef > *pixels*)

Setup the parameters for a run using the old and deeply peculiar method.

This includes the unpleasant and diffucult so use de-checkpointing code. wtf. The use of this function is very strongly deprecated.

Parameters

log_ratios	Image from which region is selected.
ims	Input data
pixels	Region for spot fitting to run in

Definition at line 1150 of file multispot5.cc.

References assert_same_size(), StateParameters::iteration, log_normal_mode(), StateParameters::pass, StateParameters::pixels, StateParameters::rng, StateParameters::spots, and spots_to_vector().

```
                                                                        {
    sort(pixels.begin(), pixels.end());

    const double variance = 1; // it should be

    //To scale the X axis of a log-normal distribution, only
    //the mu parameter needs to be changed...
```

```
const double intensity_mu = GV3::get<double>("intensity.rel_mu", 0., -1)  + l
  og(sqrt(variance));
const double intensity_sigma = GV3::get<double>("intensity.rel_sigma", 0., -1
  );
const double blur_mu = GV3::get<double>("blur.mu", 0., -1);
const double blur_sigma = GV3::get<double>("blur.sigma", 0., -1);

//The region was extracted at a certain threshold.
//These regions may be too small, so some post-region finding dilation
//may be performed. New spots are only placed at pixels which exceed the thre
  shold.
//post_dilate.threshold is (if set) used as the placing threshold so the plac
  ing threshold
//can be different from the region-finding threshold.

//Note that as a result of dliation, regions of <pixels> may be below the thr
  eshold.
//In the historic version, this could affect new spot placement. This feature
   is not supported
//in this version.
double threshold = GV3::get<double>("threshold", 0, -1);
const double post_threshold = GV3::get<double>("post_dilate.threshold", -1, 1
  );
if(post_threshold != -1)
    threshold = post_threshold;


//If dilation after region finding is to be performed, then do it here.
const double post_dilate_radius = GV3::get<double>("post_dilate.radius", 0, -
  1);
if(post_dilate_radius != 0)
{
    Image<byte> pix(ims[0].size());
    pix.fill(0);

    for(unsigned int i=0; i < pixels.size(); i++)
        pix[pixels[i]] = 255;

    Image<byte> dilated = morphology(pix, getDisc(post_dilate_radius), Morpho
  logy::BinaryDilate<byte>());

    pixels.clear();

    ImageRef p(0,0);
    do
        if(dilated[p])
            pixels.push_back(p);
    while(p.next(dilated.size()));
}


assert_same_size(ims);
if(log_ratios.size() != ims[0].size())
{
    cerr << "Bad log ratios size\n";
    exit(1);
}

vector<Vector<4> > spots;
//Spots can be either put down automatically, or specified
//The auto-initialization is very strange.
if(GV3::get<bool>("spots.auto_initialise", 1, 1))
```

```
{
    //You never get two spots in the same disc in the second stage of the alg
  orithm
    vector<ImageRef> disc = getDisc(GV3::get<double>("spot_spread", 3.1, 1));


    //Record all the pixels
    map<ImageRef, double> valid_pixels;
    for(unsigned int i=0; i < pixels.size(); i++)
        if(log_ratios[pixels[i]] > threshold)
            valid_pixels.insert(make_pair(pixels[i], log_ratios[pixels[i]]));



    //Get some initial spots by finding the local maxima
    ImageRef neighbours[8] = {
        ImageRef(-1, -1),
        ImageRef( 0, -1),
        ImageRef( 1, -1),

        ImageRef(-1,  0),
        ImageRef( 1,  0),

        ImageRef(-1,  1),
        ImageRef( 0,  1),
        ImageRef( 1,  1),
    };
    for(unsigned int i=0; i < pixels.size(); i++)
    {
        if(!(log_ratios[pixels[i]] > threshold))
            goto not_a_maximum;

        for(int j=0; j < 8; j++)
            if(!log_ratios.in_image(pixels[i] + neighbours[j]) || ! (log_rati
  os[pixels[i]] > log_ratios[pixels[i] + neighbours[j]]))
                goto not_a_maximum;

        spots.push_back(makeVector(log_normal_mode(intensity_mu, intensity_si
  gma), log_normal_mode(blur_mu, blur_sigma), pixels[i].x, pixels[i].y));

        //Remove the pixels around the initial spots
        for(unsigned int j=0; j < disc.size(); j++)
            valid_pixels.erase(pixels[i] + disc[j]);

        not_a_maximum:;
    }

    for(unsigned int i=0; i < spots.size(); i++)
        cout << spots[i] << endl;

    //Now place down extra spots in the remaining space.
    while(!valid_pixels.empty())
    {
        ImageRef p = max_element(valid_pixels.begin(), valid_pixels.end(),
  LessSecond())->first;
        spots.push_back(makeVector(log_normal_mode(intensity_mu, intensity_si
  gma), log_normal_mode(blur_mu, blur_sigma), p.x, p.y));

        for(unsigned int j=0; j < disc.size(); j++)
            valid_pixels.erase(p + disc[j]);
    }
```

```
        //This line allows extra spots to be placed down around each spot already
       put down.
        //This is a shocking hack and jenerally very unpleasant.
        double extra_r = GV3::get<double>("extra_spots", 0, 1);
        vector<ImageRef> extra = getDisc(extra_r);
        vector<Vector<4> > more_spots;
        for(unsigned int i=0; i < extra.size(); i++)
            if(extra[i] != ImageRef_zero)
                for(unsigned int j=0; j < spots.size(); j++)
                    more_spots.push_back(spots[j] + makeVector(0, 0, extra[i].x,
      extra[i].y) / (2*extra_r+1));

        copy(more_spots.begin(), more_spots.end(), back_inserter(spots));
    }
    else
    {
        Vector<>  loaded_spots = GV3::get<Vector<> >("spots.manual_spots", "", -1
      );

        if(loaded_spots.size()%4 != 0)
        {
            cerr << "Loaded spot size is not a multiple of 4\n";
            exit(1);
        }

        else
            spots = spots_to_vector(loaded_spots);
    }

    //Initialize the MT19937 RNG from a seed.
    shared_ptr<MT19937> rng(new MT19937);
    rng->simple_seed(GV3::get<int>("seed", 0, 1));

    //Load in a checkpoint (precise RNG state, iteration and pass).
    int start_iteration=0;
    int start_pass=0;
    if(GV3::get<bool>("checkpoint", 0, 1))
    {
        string rng_state = GV3::get<string>("checkpoint.rng.state", "", -1);
        istringstream rs(rng_state);
        rng->read(rs);
        start_iteration=GV3::get<int>("checkpoint.iteration", 0, -1);
        start_pass=GV3::get<int>("checkpoint.pass", 0, -1);
    }

    StateParameters p;
    p.spots = spots;
    p.rng = rng;
    p.pass = start_pass;
    p.iteration = start_iteration;
    p.pixels = pixels;

    return p;
}
```

11.11 multispot5.h File Reference

```
#include <vector>
```

```
#include <string>
#include <fstream>
#include <iostream>
#include <tr1/memory>
#include <cvd/image.h>
#include <cvd/byte.h>
#include <TooN/TooN.h>
#include <TooN/so2.h>
#include "utility.h"
#include "mt19937.h"
```

Classes

- class FitSpotsGraphics

  Graphics class for FittingSpots.
- class UserInterfaceCallback

  Callback class used by FitSpots to provide enough hooks for a user interface.
- struct UserInterfaceCallback::UserIssuedStop
- struct LogFileParseError

  Null struct thrown if a parse error is encountered when trying to load a log file.
- struct StateParameters

  Internal state (excluding fixed settings) which represents the entire internal state of spot fitting.

Functions

- std::auto_ptr< FitSpotsGraphics > null_graphics ()
- std::auto_ptr< UserInterfaceCallback > null_ui ()
- StateParameters generate_state_parameters_ye_olde (const CVD::BasicImage< double > &log_ratios, const std::vector< CVD::Image< float > > &ims, std::vector< CVD::ImageRef > pixels)
- void fit_spots_new (const std::vector< CVD::Image< float > > &ims, StateParameters &p, std::ofstream &save_spots, FitSpotsGraphics &)
- void fit_spots_new (const std::vector< CVD::Image< float > > &ims, StateParameters &p, std::ofstream &save_spots, FitSpotsGraphics &, UserInterfaceCallback &)
- StateParameters parse_log_file (std::istream &in)

11.11.1 Function Documentation

11.11.1.1 StateParameters generate_state_parameters_ye_olde (const CVD::BasicImage< double > & *log_ratios,* const std::vector< CVD::Image< float > > & *ims,* std::vector< CVD::ImageRef > *pixels*)

11.11.1.2 void fit_spots_new (const std::vector< CVD::Image< float > > & *ims,* StateParameters & *p,* std::ofstream & *save_spots,* FitSpotsGraphics &)

11.11.1.3 void fit_spots_new (const std::vector< CVD::Image< float > > & *ims,* StateParameters & *p,* std::ofstream & *save_spots,* FitSpotsGraphics & , UserInterfaceCallback &)

11.11.1.4 StateParameters parse_log_file (std::istream & *in*)

11.12 multispot5_gui.cc File Reference

FitSpots driver for interactive (GUI) operation and debugging.

```
#include <tag/printf.h>
#include <tr1/tuple>
#include <algorithm>
#include <climits>
#include <iomanip>
#include <map>
#include <cvd/image_io.h>
#include <cvd/image_convert.h>
#include <cvd/glwindow.h>
#include <cvd/morphology.h>
#include <cvd/connected_components.h>
#include <cvd/draw.h>
#include <cvd/gl_helpers.h>
#include <cvd/vector_image_ref.h>
#include <cvd/videodisplay.h>
#include <gvars3/instances.h>
#include <gvars3/GStringUtil.h>
```

```
#include <gvars3/GUI_readline.h>
#include "storm_imagery.h"
#include "multispot5.h"
#include "multispot5_place_choice.h"
#include "utility.h"
```

Classes

- class GraphicsGL

  Graphics class which draws information to the screen using OpenGL.

Functions

- double lim (double x)
- Image< byte > scale (const SubImage< double > &i, double ctr, double rng)
- void draw_bbox (const BBox &bbox)
- void watch_var (void *, string comm, string d)
- bool watch_update ()
- void GUI_Pause (int n=0)
- vector< vector< ImageRef > > get_regions (const SubImage< double > &log_ratios)
- void mmain (int argc, char **argv)
- int main (int argc, char **argv)

Variables

- map< string, string > watch

11.12.1 Detailed Description

FitSpots driver for interactive (GUI) operation and debugging.

Definition in file multispot5_gui.cc.

11.12.2 Function Documentation

11.12.2.1 double lim (double *x*)

Definition at line 34 of file multispot5_gui.cc.

Referenced by scale().

```
{
    return min(max(x, 0.), 1.);
}
```

11.12.2.2 Image<byte> scale (const SubImage< double > & *i,* double *ctr,* double *rng*)

Definition at line 40 of file multispot5_gui.cc.

References lim().

Referenced by mmain(), select_random_element(), and NegativeFreeEnergy::variance_-from_sample().

```
{
    Image<byte> s(i.size());
    for(int r=0; r < i.size().y; r++)
        for(int c=0; c < i.size().x; c++)
            Pixel::DefaultConversion<float, byte>::type::convert(lim((i[r][c] - c
      tr)/rng), s[r][c]);
    return s;
}
```

11.12.2.3 void draw_bbox (const BBox & *bbox*)

Definition at line 49 of file multispot5_gui.cc.

Referenced by GraphicsGL::draw_bbox(), GraphicsGL::draw_krap(), and mmain().

```
{
    glBegin(GL_LINES);
    glVertex(bbox.first);
    glVertex2i(bbox.first.x, bbox.first.y + bbox.second.y);

    glVertex2i(bbox.first.x, bbox.first.y + bbox.second.y);
    glVertex(bbox.first+ bbox.second);

    glVertex(bbox.first+ bbox.second);
    glVertex2i(bbox.first.x + bbox.second.x, bbox.first.y);

    glVertex2i(bbox.first.x + bbox.second.x, bbox.first.y);
    glVertex(bbox.first);

    glEnd();
}
```

11.12.2.4 void watch_var (void * , string *comm,* string *d*)

Definition at line 69 of file multispot5_gui.cc.

References watch.

Referenced by mmain().

```
{
    vector<string> vs = ChopAndUnquoteString(d);

    if(vs.size() != 1)
    {
        cerr << "Error: " << comm << " takes 1 argument: " << comm << " gvar\n";
        return;
    }

    watch[vs[0]] = GV3::get_var(vs[0]);
}
```

11.12.2.5 bool watch_update ( )

Definition at line 82 of file multispot5_gui.cc.

References watch.

Referenced by mmain().

```
{
    bool changes=0;

    for(map<string, string>::iterator i=watch.begin(); i != watch.end(); i++)
    {
        string s = GV3::get_var(i->first);

        if(s != watch[i->first])
        {
            changes=1;
            watch[i->first] = s;
        }
    }

    return changes;
}
```

11.12.2.6 void GUI_Pause (int *n* = 0)

Definition at line 100 of file multispot5_gui.cc.

```
{
    if(!GV3::get<int>("headless", 0, 1))
    {
        glFlush();
        gvar3<int> pause("pause", 1, 1);

        if(n != 0)
            *pause = n;
        (*pause)--;
        while(*pause == 0)
        {
            GUI_Widgets.process_in_crnt_thread();
            usleep(10000);
        }
```

```
    }
}
```

11.12.2.7 vector<vector<ImageRef> > get_regions (const SubImage< double > & *log_ratios*)

Definition at line 260 of file multispot5_gui.cc.

Referenced by mmain().

```
{
    gvar3<double> radius("radius", 0, 1);

    //Set the liklihood ratio threshold/spot density prior
    //same thing.
    double threshold = GV3::get<double>("threshold", 0, -1);
    int edge = GV3::get<int>("edge", 0, -1);


    //Threshold image
    Image<byte> thresholded(log_ratios.size(), 0);
    for(int r=0; r < thresholded.size().y; r++)
        for(int c=0; c < min(thresholded.size().x, edge); c++)
            thresholded[r][c] = 255 * (log_ratios[r][c] > threshold);

    //Dilate
    Image<byte> dilated = morphology(thresholded, getDisc(*radius), Morphology::B
      inaryDilate<byte>());

    transform(dilated.begin(), dilated.end(), dilated.begin(), bind1st(multiplies
      <int>(), 255));

    //Connected components of dilated image
    vector<ImageRef> fg;
    for(int r=0; r < thresholded.size().y; r++)
        for(int c=0; c < min(thresholded.size().x, edge); c++)
            if(dilated[r][c])
                fg.push_back(ImageRef(c, r));

    vector<vector<ImageRef> > regions;
    connected_components(fg, regions);

    return regions;
}
```

11.12.2.8 void mmain (int *argc*, char ** *argv*)

Definition at line 295 of file multispot5_gui.cc.

References boundingbox(), draw_bbox(), get_regions(), load_and_normalize_images(), scale(), watch_update(), and watch_var().

Referenced by main().

```
{
```

```
GUI.RegisterCommand("watch", watch_var);
GUI.LoadFile("multispot5.cfg");
int lastarg = GUI.parseArguments(argc, argv);
if(lastarg >= argc)
{
    cerr << "Specify the images to load\n";
    exit(1);
}



//Load the log_ratios image.
//We will use this as a starting point for searching for spots.
Image<double> log_ratios;
try
{
    log_ratios = img_load(GV3::get<string>("log_ratios", "", -1));
}
catch(Exceptions::All e)
{
    cerr << "Error loading " << GV3::get<string>("log_ratios", "") << ": " <<
   e.what << endl;
    exit(1);
}

Vector<2> log_ratios_zoom = GV3::get<Vector<2> >("log_ratios_zoom", "", -1);

//Load the raw data, and then load the spot parameters.
vector<string> files(argv + lastarg, argv + argc);
vector<Image<float> > ims = load_and_normalize_images(files);

//How far away from eash spot to look:
//double spot_sigmas = GV3::get<double>("sigmas", 0., -1);



gvar3<int> cluster_to_show("cluster_to_show", 0, -1);
gvar3<int> use_largest("use_largest", 0, 1);

vector<vector<ImageRef> > regions;

GLWindow win(log_ratios.size());
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_LINE_SMOOTH);

readline_in_current_thread line("> ");

gvar3<bool> start_processing("process", 0, 1);
gvar3<int> show_thresholded("show_thresholded", 0, 1);


Image<Rgb<byte> > reg(log_ratios.size(), Rgb<byte>(0,0,0));

bool first = true;
BBox cbox = make_pair(ImageRef_zero, ImageRef_zero);
for(;;)
{
    double centre = GV3::get<double>("centre", 0, -1);
    double range = GV3::get<double>("range", 0, -1);

    line.poll();
```

```
  win.make_current();
  GUI_Widgets.process_in_crnt_thread();

  if(watch_update() || first)
  {
      first = 0;

      regions = get_regions(log_ratios);

      //Colourize regions
      reg.fill(Rgb<byte>(0,0,0));
      for(unsigned int i=0; i < regions.size(); i++)
      {
          Rgb<byte> c;
          do
          {
              c.red = rand()%255;
              c.green = rand()%255;
              c.blue = rand()%255;
          }
          while(min(c.red, min(c.green, c.blue)) < 128 && min(abs(c.red - c
.green), min(abs(c.red - c.blue), abs(c.green - c.blue))) < 64);

          for(unsigned int j=0; j < regions[i].size(); j++)
              reg[regions[i][j]] = c;
      }


  }


  if(*use_largest && !regions.empty())
  {
      *cluster_to_show=0;
      for(unsigned int i=1; i < regions.size(); i++)
          if(regions[i].size() > regions[*cluster_to_show].size())
              *cluster_to_show = i;

  }
  else
      *cluster_to_show = max(min(*cluster_to_show, (int)regions.size() - 1)
, 0);

  if(!regions.empty())
      cbox = boundingbox(regions[*cluster_to_show]);

  if(!regions.empty() && *start_processing)
  {
      GraphicsGL graphics;

      place_and_fit_spots(ims, regions[*cluster_to_show], log_ratios, GV3::
get<string>("save_spots"), graphics);
      *start_processing=0;
  }

  if(*show_thresholded)
      glDrawPixels(reg);
  else
      glDrawPixels(scale(log_ratios, centre, range));

  if(cbox.second != ImageRef_zero)
      draw_bbox(cbox);
```

```
        if(win.has_events())
        {
            vector<GLWindow::Event> e;
            win.get_events(e);

            for(unsigned int i=0; i < e.size(); i++)
            {
                if(e[i].type == GLWindow::Event::RESIZE)
                {
                    ImageRef newsize = e[i].size;
                    ImageRef imsize = log_ratios.size();
                    ImageRef size=imsize;

                    float old_r = (float)imsize.x / imsize.y;
                    float new_r = (float)newsize.x / newsize.y;


                    glViewport(0, 0, newsize.x, newsize.y);
                    glMatrixMode(GL_PROJECTION);
                    glLoadIdentity();
                    double zoom;

                    if(new_r > old_r) //Then use the y axis
                        zoom = newsize.y / (float)size.y;
                    else
                        zoom = newsize.x / (float)size.x;

                    glOrtho(-.5/zoom, (newsize.x-1.5)/zoom, (newsize.y-1.5)/zoom,
       -.5/zoom, -1 , 1);

                    glPixelZoom(zoom,-zoom);
                    glRasterPos2f(0, 0);
                }
            }
        }
        win.swap_buffers();

        usleep(100000);
    }
}
```

11.12.2.9 int main (int *argc*, char ** *argv*)

Definition at line 459 of file multispot5_gui.cc.

References mmain().

```
{
    try{
        mmain(argc, argv);
    }
    catch(Exceptions::All e)
    {
        cerr << "Fatal error: " << e.what << endl;
    }
}
```

11.12.3 Variable Documentation

11.12.3.1 map<string, string> watch

Definition at line 67 of file multispot5_gui.cc.

Referenced by watch_update(), and watch_var().

11.13 multispot5_headless.cc File Reference

FitSpots driver for entierly headless (batch) operation.

```
#include <tag/printf.h>
#include <tr1/tuple>
#include <algorithm>
#include <climits>
#include <iomanip>
#include <map>
#include <cvd/image_io.h>
#include <cvd/image_convert.h>
#include <cvd/morphology.h>
#include <cvd/connected_components.h>
#include <cvd/draw.h>
#include <cvd/vector_image_ref.h>
#include <gvars3/instances.h>
#include "storm_imagery.h"
#include "multispot5.h"
#include "multispot5_place_choice.h"
#include "utility.h"
```

Functions

- vector< vector< ImageRef > > get_regions (const SubImage< double > &log_-ratios)
- void mmain (int argc, char **argv)
- int main (int argc, char **argv)

11.13.1 Detailed Description

FitSpots driver for entierly headless (batch) operation.

Definition in file multispot5_headless.cc.

11.13.2 Function Documentation

11.13.2.1 vector<vector<ImageRef> > get_regions (const SubImage< double > & *log_ratios*)

Definition at line 31 of file multispot5_headless.cc.

```
{
    gvar3<double> radius("radius", 0, 1);

    //Set the liklihood ratio threshold/spot density prior
    //same thing.
    double threshold = GV3::get<double>("threshold", 0, -1);
    int edge = GV3::get<int>("edge", 0, -1);


    //Threshold image
    Image<byte> thresholded(log_ratios.size(), 0);
    for(int r=0; r < thresholded.size().y; r++)
        for(int c=0; c < min(thresholded.size().x, edge); c++)
            thresholded[r][c] = 255 * (log_ratios[r][c] > threshold);

    //Dilate
    Image<byte> dilated = morphology(thresholded, getDisc(*radius), Morphology::B
      inaryDilate<byte>());

    transform(dilated.begin(), dilated.end(), dilated.begin(), bind1st(multiplies
      <int>(), 255));

    //Connected components of dilated image
    vector<ImageRef> fg;
    for(int r=0; r < thresholded.size().y; r++)
        for(int c=0; c < min(thresholded.size().x, edge); c++)
            if(dilated[r][c])
                fg.push_back(ImageRef(c, r));

    vector<vector<ImageRef> > regions;
    connected_components(fg, regions);

    return regions;
}
```

11.13.2.2 void mmain (int *argc*, char ** *argv*)

Definition at line 65 of file multispot5_headless.cc.

References fit_spots_new(), get_regions(), load_and_normalize_images(), null_graphics(), open_or_die(), parse_log_file(), and LogFileParseError::what.

```
{
    GUI.LoadFile("multispot5.cfg");
    int lastarg = GUI.parseArguments(argc, argv);
    if(lastarg >= argc)
    {
        cerr << "Specify the images to load\n";
        exit(1);
    }
    vector<string> files(argv + lastarg, argv + argc);

    //Save this now since the de-checkpointing code will kl0bber it
    //when it reloads the gvars
    string save_spots_file = GV3::get<string>("save_spots", "", -1);

    string checkpoint_file = GV3::get<string>("load_checkpoint", "", 1);

    if(checkpoint_file != "")
    {
        //Load and de-checkpointing
        ifstream chk;
        open_or_die(chk, checkpoint_file);

        StateParameters p;

        try{
            p = parse_log_file(chk);
        }
        catch(LogFileParseError e)
        {
            cerr << "SI TEH FUX0R11ONEone!oneleven: " << e.what << endl;
            exit(1);
        }

        vector<Image<float> > ims = load_and_normalize_images(files);

        //Restore kl0bbered variable
        GV3::get<string>("save_spots") = save_spots_file;

        ofstream save_spots;
        open_or_die(save_spots, save_spots_file);

        fit_spots_new(ims, p, save_spots, *null_graphics());

    }

    vector<Image<float> > ims = load_and_normalize_images(files);

    //Load the log_ratios image.
    //We will use this as a starting point for searching for spots.
    Image<double> log_ratios;
    try
    {
        log_ratios = img_load(GV3::get<string>("log_ratios", "", -1));
    }
    catch(Exceptions::All e)
    {
        cerr << "Error loading " << GV3::get<string>("log_ratios", "") << ": " <<
       e.what << endl;
        exit(1);
    }
```

```
    gvar3<int> cluster_to_show("cluster_to_show", 0, -1);
    gvar3<int> use_largest("use_largest", 0, 1);

    vector<vector<ImageRef> > regions;

    regions = get_regions(log_ratios);
    if(regions.size() == 0)
    {
        cerr << "There are no regions!\n";

        ofstream save_spots;
        open_or_die(save_spots, save_spots_file);
        save_spots << "NOREGIONS\n";

        exit(1);
    }

    if(*use_largest && !regions.empty())
    {
        *cluster_to_show=0;
        for(unsigned int i=1; i < regions.size(); i++)
            if(regions[i].size() > regions[*cluster_to_show].size())
                *cluster_to_show = i;

    }
    else
        *cluster_to_show = max(min(*cluster_to_show, (int)regions.size() - 1), 0)
      ;


    auto_ptr<FitSpotsGraphics> gr = null_graphics();
    place_and_fit_spots(ims, regions[*cluster_to_show], log_ratios, save_spots_fi
      le, *gr);
}
```

11.13.2.3 int main (int *argc*, char ** *argv*)

Definition at line 161 of file multispot5_headless.cc.

References mmain().

```
{
    try{
        mmain(argc, argv);
    }
    catch(Exceptions::All e)
    {
        cerr << "Fatal error: " << e.what << endl;
    }
}
```

11.14 multispot5_jni.cc File Reference

```
#include <sstream>
```

```
#include <algorithm>
#include <cvd/image.h>
#include <cvd/image_convert.h>
#include "ThreeBRunner.h"
#include "storm_imagery.h"
#include "multispot5.h"
#include "multispot5_place_choice.h"
#include "utility.h"
#include <gvars3/instances.h>
#include <tag/printf.h>
#include <tr1/tuple>
```

Classes

- class JNIUserInterface

  3B User interface for the Java plugin.

Functions

- string get_string (JNIEnv *env, jstring js)
- Image< jbyte > get_local_copy_of_image (JNIEnv *env, jbyteArray data, int rows, int cols)
- Image< float > get_local_copy_of_image (JNIEnv *env, jfloatArray data, int rows, int cols)
- JNIEXPORT void JNICALL Java_ThreeBRunner_call (JNIEnv *env, jobject jthis, jstring cfg, jobjectArray images, jbyteArray mask_data, jint n_images, jint rows, jint cols, jstring file)

11.15 randomc.h File Reference

Defines

- #define INT64_SUPPORTED

Typedefs

- typedef signed int int32_t
- typedef unsigned int uint32_t
- typedef long long int64_t
- typedef unsigned long long uint64_t

Functions

- void EndOfProgram (void)
- void FatalError (const char *ErrorText)

11.15.1 Detailed Description

Definition in file randomc.h.

11.15.2 Define Documentation

11.15.2.1 #define INT64_SUPPORTED

Definition at line 122 of file randomc.h.

11.15.3 Typedef Documentation

11.15.3.1 typedef signed int int32_t

Definition at line 118 of file randomc.h.

11.15.3.2 typedef unsigned int uint32_t

Definition at line 119 of file randomc.h.

11.15.3.3 typedef long long int64_t

Definition at line 120 of file randomc.h.

11.15.3.4 typedef unsigned long long uint64_t

Definition at line 121 of file randomc.h.

11.15.4 Function Documentation

11.15.4.1 void EndOfProgram (void)

11.15.4.2 void FatalError (const char * *ErrorText*)

11.16 sampled_multispot.h File Reference

```
#include <vector>
#include <cvd/image_ref.h>
#include <tr1/tuple>
#include <TooN/TooN.h>
#include "drift.h"
#include "spot_with_background.hh"
```

Classes

- struct SampledMultispot::SpotWithBackgroundMasked

  This class compute the log-diff-hess probability of a spot, given an image patch and background due to existing spots.
- struct SampledMultispot::SpotWithBackgroundMasked

  This class compute the log-diff-hess probability of a spot, given an image patch and background due to existing spots.
- class SampledMultispot::GibbsSampler

  Draw samples from the spot states given the spots positions and some data.
- class SampledMultispot::GibbsSampler2

  Gibbs sampling class which masks spots to reduce computation.

Namespaces

- namespace SampledMultispot

Defines

- #define SWBG_NAME SpotWithBackground
- #define SWBG_NAME SpotWithBackgroundMasked
- #define SWBG_HAVE_MASK

Typedefs

- typedef char State

Functions

- double SampledMultispot::intensity (double i)
- double SampledMultispot::intensity (const pair< double, Vector< 4 > > &i)
- template<class T >
  void SampledMultispot::remove_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample)
- template<class T >
  void SampledMultispot::add_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample)
- template<class T >
  void SampledMultispot::remove_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- template<class T >
  void SampledMultispot::add_spot (vector< vector< double > > &current_sample_intensities, const vector< T > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- template<class T >
  void SampledMultispot::remove_spot (vector< vector< double > > &current_sample_intensities, const vector< vector< T > > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- template<class T >
  void SampledMultispot::add_spot (vector< vector< double > > &current_sample_intensities, const vector< vector< T > > &spot_intensities, const vector< State > &spot_sample, const vector< int > &mask)
- vector< double > SampledMultispot::compute_spot_intensity (const vector< ImageRef > &pixels, const Vector< 4 > &params)
- vector< pair< double, Vector< 4 > > > SampledMultispot::compute_spot_intensity_derivatives (const vector< ImageRef > &pixels, const Vector< 4 > &params)
- vector< tuple< double, Vector< 4 >, Matrix< 4 > > > SampledMultispot::compute_spot_intensity_hessian (const vector< ImageRef > &pixels, const Vector< 4 > &params)
- vector< int > SampledMultispot::sequence (int n)

11.16.1 Define Documentation

11.16.1.1 #define SWBG_NAME SpotWithBackground

Definition at line 29 of file sampled_multispot.h.

11.16.1.2 #define SWBG_NAME SpotWithBackgroundMasked

Definition at line 29 of file sampled_multispot.h.

11.16.1.3 #define SWBG_HAVE_MASK

Definition at line 30 of file sampled_multispot.h.

11.16.2 Typedef Documentation

11.16.2.1 typedef char State

Definition at line 11 of file sampled_multispot.h.

11.17 storm.h File Reference

Code dealing with storm imagery (high level).

```
#include <TooN/TooN.h>
#include <cvd/image.h>
#include <utility>
#include <tr1/tuple>
#include "utility.h"
```

Functions

- template<class B >
  double spot_shape_s (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::pair< double, TooN::Vector< 4 > > spot_shape_diff_position (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::tr1::tuple< double, TooN::Vector< 4 >, TooN::Matrix< 4 > > spot_shape_hess_position (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::tr1::tuple< double, TooN::Vector< 2 >, TooN::Matrix< 2 > > spot_shape_hess (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  std::pair< double, TooN::Vector< 2 > > spot_shape_diff (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- template<class B >
  double spot_shape (const TooN::Vector< 2 > &x, const TooN::Vector< 4, double, B > &phi)
- double log_probability_no_spot (const CVD::SubImage< float > &im, double variance)

- template<class Base >
  std::tr1::tuple< double, TooN::Vector< 2 >, TooN::Matrix< 2 > > log_probability_spot_hess (const CVD::SubImage< float > &im, double variance, const TooN::Vector< 4, double, Base > &spot_parameters)
- template<class Base >
  std::pair< double, TooN::Vector< 2 > > log_probability_spot_diff (const CVD::SubImage< float > &im, double variance, const TooN::Vector< 4, double, Base > &spot_parameters)
- template<class Base >
  double log_probability_spot (const CVD::SubImage< float > &im, double variance, const TooN::Vector< 4, double, Base > &spot_parameters)
- double log_normal_std (double mu, double sigma)
- double log_normal_mode (double mu, double sigma)
- double log_log_normal (double x, double mu, double sigma)
- double diff_log_log_normal (double x, double mu, double sigma)
- double hess_log_log_normal (double x, double mu, double sigma)

11.17.1 Detailed Description

Code dealing with storm imagery (high level).

Definition in file storm.h.

11.17.2 Function Documentation

11.17.2.1 double log_probability_no_spot (const CVD::SubImage< float > & *im*, double *variance*) [inline]

Find the log probability of an image patch, assuming zero mean and the given variance, and no spot present.

See also log_probability_spot()

Parameters

im	Image
variance	variance

Returns

The log probability

Definition at line 183 of file storm.h.

```
{
    double logprob_part=0;
    for(int y=0; y < im.size().y; y++)
        for(int x=0; x < im.size().x; x++)
            logprob_part -= im[y][x] * im[y][x];
```

```
    return logprob_part/(2*variance) - im.size().area() * log(2*M_PI*variance)/2;

}
```

11.18 storm_imagery.cc File Reference

Code dealing with storm imagery (low level).

```
#include <gvars3/instances.h>
#include <cvd/image_io.h>
#include <cvd/convolution.h>
#include <TooN/wls.h>
#include <tr1/tuple>
#include "storm_imagery.h"
#include "debug.h"
#include "utility.h"
```

Functions

- vector< Image< float > > load_and_preprocess_images2 (const vector< string > &names)
- vector< Image< float > > load_and_preprocess_images (const vector< string > &names)
- pair< double, double > auto_fixed_scaling (const vector< Image< float > > &ims, double frac)
- vector< Image< float > > load_and_normalize_images (const vector< string > &files)

11.18.1 Detailed Description

Code dealing with storm imagery (low level).

Definition in file storm_imagery.cc.

11.18.2 Function Documentation

11.18.2.1 pair<double, double> auto_fixed_scaling (const vector< Image< float > > & *ims*, double *frac*)

Compute the mean and variance of the (on average) darkest pixels, in order to find the correct scaling, by examining hte background.

Definition at line 197 of file storm_imagery.cc.

References assert_same_size(), and sq().

Referenced by load_and_normalize_images().

```
{
    assert_same_size(ims);

    //Compute the mean image (ish)
    Image<double> ave(ims[0].size());
    ave.fill(0);
    for(unsigned int i=0; i < ims.size(); i++)
        for(int y=0; y < ave.size().y; y++)
            for(int x=0; x < ave.size().x; x++)
                ave[y][x] += ims[i][y][x];

    //Find the smallest N% of the pixels...
    vector<pair<double, ImageRef> > pixels;
    for(int y=0; y < ave.size().y; y++)
        for(int x=0; x < ave.size().x; x++)
            pixels.push_back(make_pair(ave[y][x], ImageRef(x,y)));

    int npix = (int) floor(frac *pixels.size() + 0.5);
    npix = max(0, min(npix, (int) pixels.size()));

    nth_element(pixels.begin(), pixels.begin() + npix, pixels.end());

    pixels.resize(npix);

    //Now compute the mean and variance of those pixels.
    double sum=0, sum2=0;

    for(unsigned int i=0; i < ims.size(); i++)
    {
        for(unsigned int j=0; j < pixels.size(); j++)
        {
            sum += ims[i][pixels[j].second];
            sum2 += sq(ims[i][pixels[j].second]);
        }
    }

    double num = 1.0 * pixels.size() * ims.size();
    double mean = sum / num;
    double std  = sqrt(((sum2/num) - mean*mean) * num / (num-1));

    cout << "Automatic determination of fixed scaling:" << endl
         << "mean       = " << mean << endl
         << "std        = " << std << endl
         << "sqrt(mean) = " << sqrt(mean*255)/255 << endl;

    return make_pair(mean, std);
}
```

11.19 storm_imagery.h File Reference

Code dealing with storm imagery (low level).

```
#include <cvd/image.h>
#include <cvd/byte.h>
#include <utility>
#include <vector>
```

11.19.1 Detailed Description

Code dealing with storm imagery (low level).

Definition in file storm_imagery.h.

11.20 three_B.java File Reference

Classes

- class ThreeBGlobalConstants
- class SPair

  Utility class to hold a pair of strings.
- class Util

  Utility calss to hold a number of handy static functions.
- class three_B

  ImageJ plugin class.
- class AdvancedDialog

  Control panel which basically presents the .cfg file in a large edit box along with additional necessary things (frame range and pixel size).
- class ThreeBDialog

  Dialog box for starting 3B The dialog highlights bad things in red.
- class Spot

  Basic spot class, simply contains coordinates.
- class SomethingChanges

  Listener class which triggers a complete redraw.
- class FloatSliderWithBox

  This class makes a floating point slider bar with an edit box next to it for more precision.
- class FloatSliderWithBox::SliderChanged
- class FloatSliderWithBox::TextChanged
- class CloseButtonListener

  Close button issues a window close event.
- class StopButtonListener

  Stop 3B thread.

- class ExportButtonListener

  Listener for the export button.
- class EControlPanel

  Control panel for running 3B plugin and providing interactive update.
- class ThreeBRunner

  This class deals with running the actual 3B code It should be run in a separate thread It will make calls back to a viewer, and accepts termination calls as well.

11.21 ThreeBHelp.java File Reference

Classes

- class ThreeBHelp

  3B plugin cclass to bring up a basic help window.

11.22 ThreeBLoader.java File Reference

Classes

- class ThreeBLoader

  Plugin class to load up an old 3B run.

11.23 utility.cc File Reference

Utility bits.

```
#include "utility.h"
#include "debug.h"
#include <climits>
```

11.23.1 Detailed Description

Utility bits.

Definition in file utility.cc.

11.24 utility.h File Reference

Utility bits.

```
#include <cvd/image.h>
#include <vector>
#include <string>
#include <cstring>
#include <cerrno>
#include <cstdlib>
#include <utility>
```

Typedefs

- typedef std::pair< CVD::ImageRef, CVD::ImageRef > BBox

Functions

- double sign (double x)
- float sq (float f)
- double sq (double f)
- const std::vector< CVD::SubImage< float > > sub_images (const std::vector< CVD::Image< float > > &im, CVD::ImageRef pos, CVD::ImageRef size)
- std::pair< CVD::ImageRef, CVD::ImageRef > boundingbox (const std::vector< CVD::ImageRef > &all_spots)
- template<class Stream >
  void open_or_die (Stream &save_spots, const std::string &save_spots_file)

11.24.1 Detailed Description

Utility bits.

Definition in file utility.h.

11.24.2 Function Documentation

11.24.2.1 double sq (double *f*) [inline]

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Definition at line 34 of file utility.h.

```
{ return f*f; }
```