Software Engineering Project "Bugfind"
Testing Document

Project group
Perttu Haimi, Petteri Hintsanen, Philipp Masche, Toni Ruokolainen and Osku Salerma

Instructor
Peter Lamberg

Helsinki 9th May 2001
Testing Document
UNIVERSITY OF HELSINKI
Department of Computer Science

Version history

- v0.1, 25.3.2001 (first draft)
- v0.2, 22.4.2001 (test case specifications almost done)
- v0.3, 29.4.2001 (minor corrections)
- v1.0, 9.5.2001 (Final version)

Contents

1 Preface

This paper contains the detailed testing plan and documentation of the Bugfind program. It outlines how the different modules were tested and how the integration testing was done. This paper is mainly based on the modules described in the design specification document.

All the primitive data types used in the following text are more or less the ones used in the C programming language. That is, `unsigned char` means an unsigned byte value, `double` means a double-precision floating point value, `int` means an integer value, and so on.

2 Module-specific test cases

The module test cases contain a test description, expected result and the type of the test. If test is very simple, no test results are documented, since all test cases have been run succesfully. For more complicated tests there may be also a description of the results.

2.1 file.c

- **file 1**: Loading a file (both pgm and mrc) which does not exist.
  Expected result: Error message
  Test type: manual

- **file 2**: Loading a pgm-image, saving as mrc, converting back to pgm
  Expected result: Similar picture, no complaints from the pgm-viewer.
  Test type: manual

- **file 3**: Loading 16 bit pgm-image
  Expected result: Error message
  Test type: manual

- **file 4**: Loading mrc-image, saving as pgm, opening with xview/gimp.
  Expected result: Similar picture, no complaints from the viewer.
  Test type: manual

2.2 gaussian.c

- **gaussian 1**: radius <= 0
  Parameters: image and the radius
  Expected result: Error message
  Test type: manual

- **gaussian 2**: radius > image size
  Parameters: image and the radius
  Expected result: Image filtered with large radius setting.
  Test type: manual

- **gaussian 3**: NULL image
  Expected result: Error message
  Test type: manual

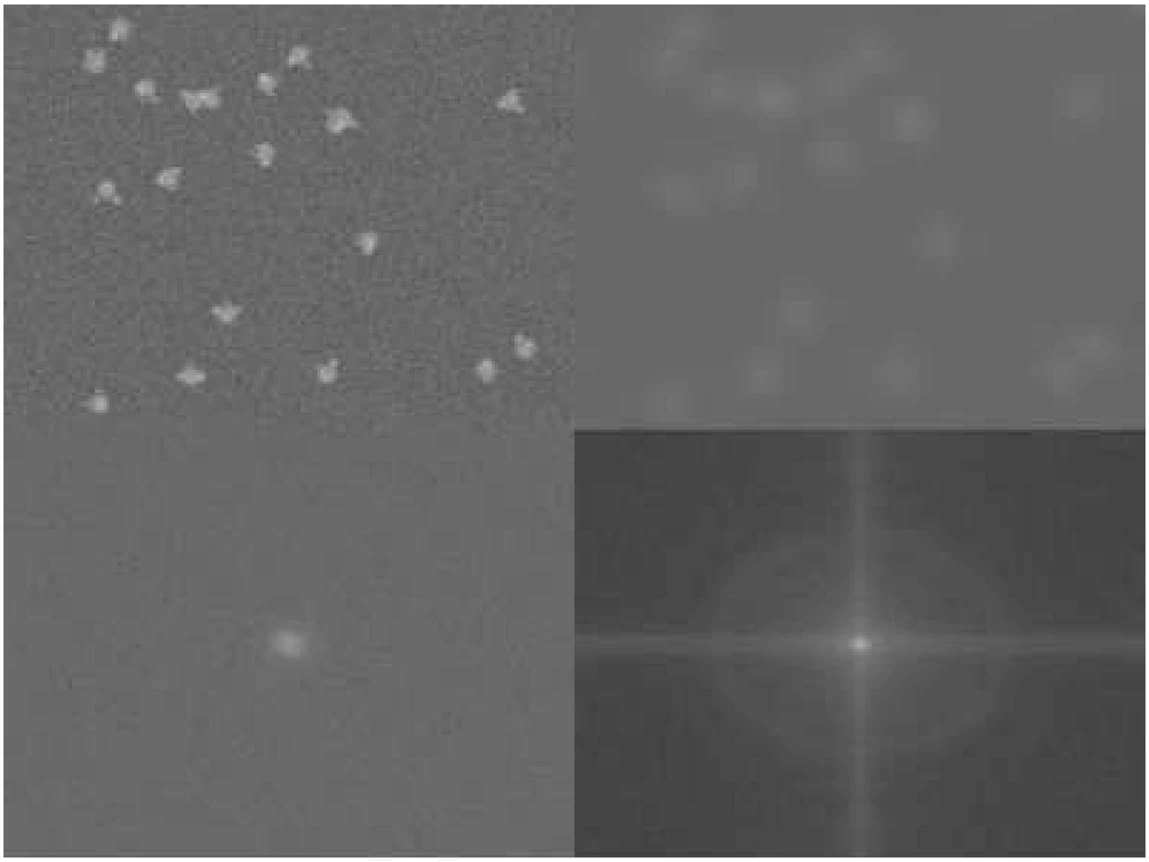

Figure 1: The effect of gaussian filtering to fourier power spectrum

- **gaussian 4**: Making a fourier (power) spectrum of filtered image.
  Expected result: The high frequencies should be smoothly attenuated.
  Test type: manual
  Description of results: In figure 1 in upper left corner there is an artificial noisy image, upper left is gaussian filtered image (with gaussian implementation from gimp), and under each image there is a 2D fourier power spectrum of the image. In the filtered image the low frequences are more intensive than in the non-filtered image. There is still some high frequences left, one cause may be that the filtered image had very few gray values, since it was truncated to 8 bits. Our own version of gaussian gave similar results. Matlab commands for producing the fourier transforms are below.

```
[orig, map] = imread('input.bmp');
trans = fftshift(fft2(orig));
power = log(abs(trans));
scaled = (power-min(min(power)))*(255/(max(max(power))-min(min(power))));
imwrite(scaled, map, 'output.bmp');
```

- **gaussian 5**: A test image with only one pixel set to 255 at the middle.
  Expected result: The image should have smooth transition with gaussian distribution with the radius of the filter.
  Test type: manual
  Description of results: In figures 2 and 3 there is the profile of the filterd image when

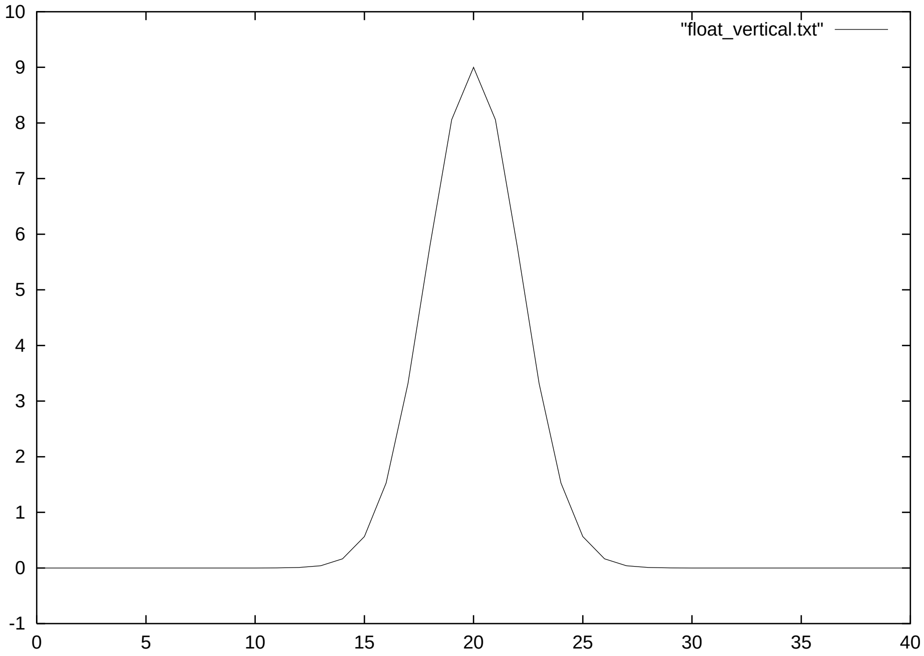

Figure 2: Profile of floating point version (modified from gimp) of gaussian filtering

taking the values of the column which has the point set to 255. The horisontal direction gives the same results. The width parameter was set to five, because the averaging effect of gaussian makes intensities in this kind of image very small, causing them to be truncated to zero in the integer version. We also coded our own implementation of gaussian filtering, and it gave the same results as the integer version of gimp gaussian.

- **gaussian 6**: Visual inspection of filtered image with various radius settings
  Expected result: No artifacts at borders, amount of blurring consistent with the radius.
  Test type: manual

2.3 findmax.c

- **findmax 1**: One maximum with known coordinate (0,0)
  Expected result: Exactly one result with coordinate (0,0)
  Test type: automatic

- **findmax 2**: No maxima in image
  Expected result: Maxima should be reported to (9,9)
  Test type: automatic

- **findmax 3**: Double maxima in coordinates (0,0) and (19,19)
  Expected result: Maxima should be reported to (9,9)
  Test type: automatic

- **findmax 4**: Triple maxima in coordinates (0,0), (0,19) and (19,19)
  Expected result: Maxima should be reported to (12,6).
  Test type: automatic

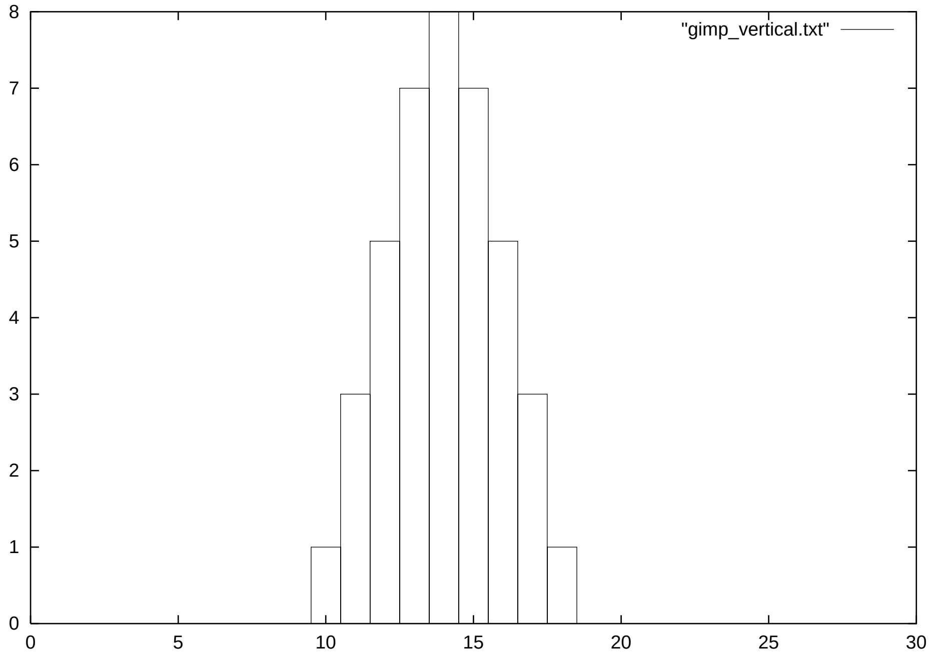

Figure 3: Profile of integer version (gimp) of gaussian filtering

- **findmax 5**: Quadruple maxima in coordinates (0,0), (0,19), (0,19) and (19,19)
  Expected result: Maxima should be reported to (9,9).
  Test type: automatic

- **findmax 6**: All values are maxima
  Expected result: Maxima should be reported to (9,9).
  Test type: automatic

- **findmax 7**: Box diamater < 0
  Expected result: No maxima should be reported.
  Test type: automatic

- **findmax 8**: Box diamater > width of the image
  Expected result: Two maxima should be reported with known values, because two partical boxes will be used.
  Test type: automatic

- **findmax 9**: Box diamater > height of the image
  Expected result: Two maxima should be reported with known values, because two partical boxes will be used.
  Test type: automatic

- **findmax 10**: Box diamater > height & width of the image
  Expected result: One maximum should be reported with known value.
  Test type: automatic

Test *findmax 7* resulted in a crash. The bug is now removed and new implementation is tested.

2.4 comparison.c

- **comparison 1**: Compare image to known coordinate in bigger image.
  Expected result: Correlation of 1.0 in known coordinates
  Test type: automatic

- **comparison 2**: Compare inverted image to known coordinate in bigger image.
  Expected result: Correlation of -1.0 in known coordinates
  Test type: automatic

- **comparison 3**: Compare blank image to known coordinate in bigger image.
  Expected result: Correlation of -100.0 in coordinates (0,0)
  Test type: automatic

- **comparison 4**: Compare image to known coordinate in bigger image which is blank.
  Expected result: Correlation of -100.0 in coordinates (0,0)
  Test type: automatic

2.5 rotation.c

- **rotation 1**: Test rotation angle system by rotating an image by $\pm 2\pi$.
  Expected result: Identical images.
  Test type: Automatic.

- **rotation 2**: Test rotation quality by first rotating an image by α degrees and then rotating the resulting image back to the original by $-\alpha$ degrees.
  Expected result: Exact match.
  Test type: Automatic.

- **rotation 3**: Rotate trivial images, like straight lines, circles, etc... by 90 degrees ($\pi/2$) and 270 degrees ($3\pi/2$). Compare the results against preproduced images. *Expected result*: Exact matches.
  Test type: Automatic.

2.6 projection.c

- **projection 1**: Test projection angle system by projecting a premade model from some angle (α, β) and from the angles producing the same result with some additional in-plane rotations if required. Thus the projection test also tests the rotation module's point-rotating functions thoroughly. Finally compare the results.
  Expected result: Exact matches.
  Test type: Automatic.

2.7 scaling.c

- **scaling 1**: Scale a test image by shrinking to a half of the size of the original. Save the results for visual inspection.
  Expected result: Scaled image of size 360×240, looking somewhat like the expected output `scaling_1_expected.png`.
  Test type: Manual.

- **scaling 2**: Scale a test image by magnifying it by a factor of 1.5. Save the results for visual inspection.
  Expected result: Scaled image of size 720×480, looking somewhat like the expected output `scaling_2_expected.png`.
  Test type: Manual.

2.8 interpolation.c

- **interpolation 1**: Test nearest-neighbour interpolation with a small 5x5 gradient pattern by sweeping over the pattern diagonally using floating-point pixel values with an interval of 0.1. Also use negative and out-of-bounds coordinate value. Compare the results against precalculated intensities.
  Expected result: Exact matches.
  Test type: Automatic.

- **interpolation 2**: Test linear interpolation by creating a simple linear gradient by streching two pixels ("black and white") into five pixels (should produce gradient-like pattern). Also use negative and out-of-bounds values. Compare the result against a precalculated pattern.
  Expected result: Exact matches.
  Test type: Automatic.

2.9 config.c

- **config 1**: The configuration file contains totally wrong data.
  Expected result: Error message.
  Test type: manual

- **config 2**: The configuration file does not exist.
  Expected result: Error message.
  Test type: manual

- **config 3**: An expected integer parameter is not a integer value. This test applies for all different integer parameters.
  Expected result: Error message.
  Test type: manual

- **config 4**: An expected float/double parameter is not a float/double value. This test applies for all different float/double parameters.
  Expected result: Error message.
  Test type: manual

- **config 5**: An integer parameter is out of range. This test applies for all different integer parameters.
  Expected result: Error message.
  Test type: manual

- **config 6**: A float/double parameter is out of range. This test applies for all different float/double parameters.

Expected result: Error message.
Test type: manual

- **config 7**: A parameter does not contain any value. This test applies for all different parameters.
  Expected result: Error message.
  Test type: manual

- **config 8**: The configuration option is not known.
  Expected result:Ignoring this option.
  Test type: manual

- **config 9**: The configuration option has already occured once before in the file.
  Expected result: Ignoring this option.
  Test type: manual

2.10 bf_mpi.c

- Sending and recieving small and large files

- Starting and terminating

2.11 misc.c

- **misc 1**: More memory is tried to be allocated than available.
  Expected result: Error message.
  Test type: manual

- **misc 2**: Test if memory is allocated and freed propper.
  Expected result: No special reaction by the program.
  Test type: manual

- **misc 3**: Test if panic is shutting down everythink fine. This test has to be run from all different program states, in which panic could be called.
  Expected result: all processes of the program shut down. Error Message.
  Test type: manual

3 Integration testing

Since we have a fairly small number of modules, we decided to do integration testing in one step ie. to test the program with all modules in place. Also, the various utilities that were made to help testing use many modules. Namely, image conversion utilities use file and image modules, test image generator uses also rotation, projection and interpolation. This gave us confidence that the number of bugs would not be so large that it would prevent efficient testing.

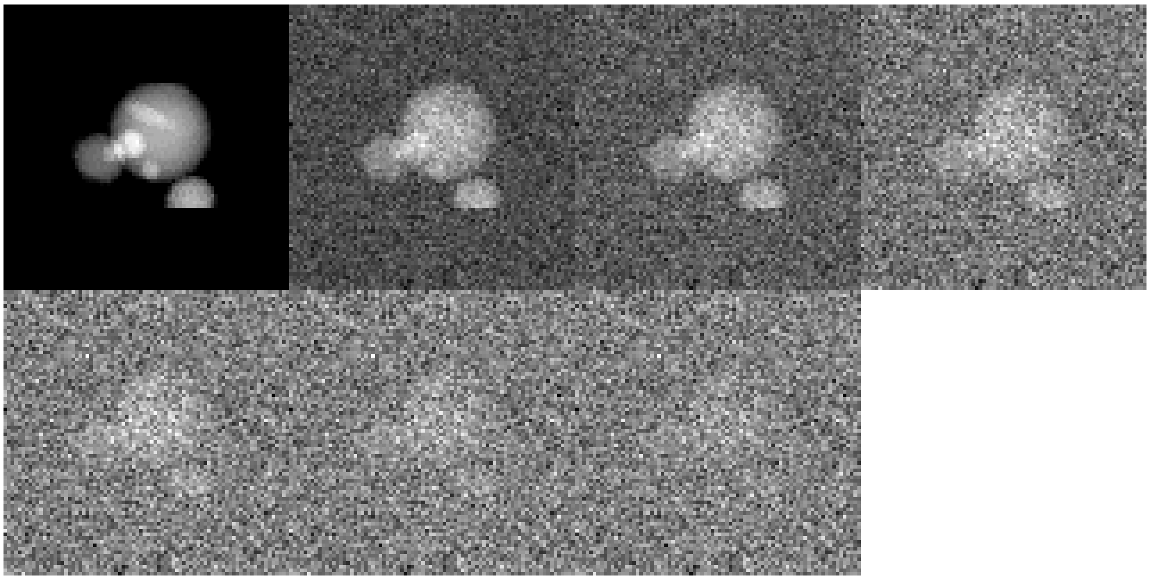

Figure 4: Images used in the noise test

4 Validation testing

The validation test cases contain a unformal test description to give an idea, what was how tested.

- **Rod1:** An easy artificial model (rod) is randomly projected in an image. Then noise is added. Bugfind finds these particles with a correlation value around 0.9.

- **Rod2:** An easy artificial model (rod) is randomly projected in an image, noise is added, the image is scaled down. Bugfind finds the particles still quite good.

- **3Dcross:** A 3D cross with center of mass not in the center of the model is projected randomly to an image, then noise is added. Bugfins does not find all particles, the correlation values are 0.3 and less.

- **Ribosome:** The real-data ribosome model is randomly projected to an image, then noise is added. Bugfind finds 10 of 15 particles with correlation values between 0.5 and 0.3.

- **Output angles:** Visual comparison of output angles to original values of the angles in artificial images. This test gave bad results, and the reason appeared to be that test generator (testgen) which makes the projections with modified projection module, which uses all three angles, whereas bugdfind projects by setting the third angle to zero and later rotating the 2D image in plane. These angle systems are not comparable.

- **Effect of noise:** Small images (see figure 4) with varying noise levels. The results are in table 1.

- **Scaletest:** The same image used in the Ribosome test is scaled down to 78% of the original size. Bugfind finds only 8 particles, the correlation values are between 0.7 and 0.45.

SNR	Best match
infinity	0.899113
2	0.822357
1	0.761861
0.5	0.672393
0.25	0.561756
0.125	0.444860
0.0625	0.337230
0.03125	0.256273

Table 1: Effect of noise

- **Spherical virus image and model:** A 2000×2000 part of the image d450.mrc was used as the image, lp.mrc was used as model. The number of nodes was 7 (master + 6 slave nodes). The density setting was 30, diameter 360 and searchradius 0. With these settings there is 36 candidate points, and doing the comparisons took approximately 10 minutes. The correlation values were very low, less than 0.26. It seems that bugfind is very dependent on the quality of candidate points. This testimage had several virus particles very close to each other, and that may be the reason gaussian filter did not work very well.

- **Pizza:** One pizza taken from a real micrograph and doing comparison with model *lp.mrc* on each point of micrograph. The model was projected only from angle (0,0) and different scaling values were used. Results were not good: the best correlation values were between (0.04 - 0.055) and the middle point was reported somewhere else it should be. This could imply that the comparison code using Pearson's correlation doesn't work well with this kind of data. With hand-generated data with high contrast and quite low noise-ratio (images were generated with Gimp) the comparison-code reported the match as expected.