Verification and Validation Plan for ImgBeamer

Joachim de Fourestier

March 27, 2023

1 Revision History

Date	Version	Notes
2023/02/17	1.0	First version
2023/02/20	1.0.1	fix minor issues in T2, T3, T16, and T18
	1.0.1	Minor issues in general information
	1.0.2	Fix grammar in sections 4.2, 4.3, 4.5, and 4.6.
	1.0.3	Fix formatting and grammar in sections 5, 5.1.1, 5.1.2, and 5.1.3.
	1.0.4	Fix grammar in Section 5.2.
	1.0.5	Tweaks to Section 3.2.
	1.0.6	Added missing references to MG and MIS.
2023/02/22	1.0.7	Provide specific values for most of the tests.
	1.0.8	Review the design and VnV verification plans.
2023/02/27	1.0.9	Improve wording for Section 4.5.
	1.0.10	Tweak tests in sections 5.1.2 and 5.1.3.
2023/03/23	1.0.11	Improve clarity for how some tests will be performed.
2023/03/24	1.1.0	Incorporate feedback from Dr. Smith's review.
	1.1.1	Added a code linting ruleset.
	1.1.2	Added symbolic constants.
2023/03/25	1.2.0	Fill in parts of the unit testing (section 6).
	1.2.1	Added details in the Appendix (8).
2023/03/26	1.2.2	Fill in expected outputs and justifications for the unit tests in section 6.2 and 6.3.
2023/03/27	1.2.3	Added web link pointing to unit test files.

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

symbol	description
GUI	Graphical User Interface
MG	Module Guide
MIS	Module Interface Specification
MSE	Mean Squared Error
MS-SSIM	Multi-scale Structural Similarity
NFR	Nonfunctional Requirement
NRMSE	Normalized Root Mean Square Error
PSNR	Peak Signal-to-Noise Ratio
R	Requirement
RMSE	Root Mean Square Error
ROI	Region of Interest
SCC	Spatial Correlation Coefficient
SRS	Software Requirements Specification
SSIM	Structural Similarity
T	Test
U	Unit test
UI	User Interface
UIQ	Universal Image Quality
VIF	Visual Information Fidelity
VnV	Verification and Validation

3 General Information

This document lays out the Verification and Validation (VnV) plan of ImgBeamer as described by the Software Requirements Specification (SRS) [5]. Testing of the software and its components will be conducted to build confidence in its usability, accuracy, and, ultimately, whether it meets the requirements from the SRS.

3.1 Summary

ImgBeamer software aims to be a demonstration and learning tool of how a scanning electron microscope (SEM) formulates an image. The idea is to help visualize the influence trends of imaging parameters (as defined in the SRS) on image quality (or clarity) and resolution.

3.2 Objectives

The objective is to produce a tool that is easy to use (*usability*). It should feel intuitive to the user and should provide easy-to-understand information. This is so the user may identify any trends in how the final image is affected with relative ease and confidence. As a secondary goal, the *accuracy* in the trends are important. Due to the subjective and qualitative nature of evaluating image quality, the metric used should be straight-forward and provide some objectivity (for quantitative comparison) and assurance to the users. As for an objective that will be left out, due to resource limitations, is the study and improvement of an all-round robust (quantitative and consistent) image metric. While this is a highly desired goal, this is an incredibly difficult task and is currently deemed to be outside the scope of this project.

3.3 Relevant Documentation

There are multiple design documents that provide the important and intimate details to understand some of the concepts that are being tested. These documents are the following:

- Problem Statement [4]: States the problem we are trying to solve and introduces the topics that will need be verified.

- SRS [5]: States the requirements, assumptions, models, and important terminology that are used in the VnV plan.
- VnV Report [6]: States what has been achieved from the VnV plan, as well as provide results or evidence to help build confidence for correctness.
- MG [2]: States the responsibilities and secrets for each module that will be evaluated by the VnV plan.
- MIS [3]: Specifies the functional design and states what is needed for each module that will be evaluated by the VnV plan.

4 Plan

In this section, multiple plans are described to test and inspect the software with an emphasis on *usability*. Multiple approaches and perspectives will be employed by the VnV team (4.1) to help build confidence in the requirements, to avoid any missed important details, and to deliver on the qualities as mentioned in the objective (3.2).

4.1 Verification and Validation Team

The members of the VnV team as well their individual roles are listed in the following table:

Role	Name
Project Supervisor	Dr. Spencer Smith
Author	Joachim de Fourestier
Domain Expert	Karen Wang
SRS Reviewer	Jason Balaci
VnV Plan Reviewer	Sam Crawford
MG + MIS Reviewer	Lesley Wheat
Expert Consultant	Dr. Nabil Bassim
Expert Consultant	Michael W. Phaneuf

Table 1: Table of the VnV Team Members

Note: Dr. Bassim is my PhD Supervisor, and Mr. Phaneuf is my co-supervisor (employer/industry), for an industrial PhD. Much of this project was originally conceptualized with the guidance and help of Mr. Phaneuf. This is acknowledged here.

4.2 SRS Verification Plan

The SRS will be reviewed by the project supervisor, the SRS reviewer and the author. Some input may be given by the expert consultants if time permits. Most of the feedback has been provided as issues on GitHub, as annotated documents, or by verbal exchange. Throughout the development of the project, the author is expected to make changes needed to the resolve the issues. The reviewers may refer to the SRS checklist [11]. The key objective is to verify that the software requirements and the documentation are sound and coherent to the intended audience as defined in the SRS.

4.3 Design Verification Plan

Much of the conceptualization was done after having multiple discussions with the expert consultants and having done literature review for preexist-

ing tools and documentation relevant to the project. Decisions were made with a focus on the usability, meaning little to no setup required. The design and implementation is documented in the MG[2]/MIS[3] which will be verified with the MG [9] and MIS [10] checklists. The VnV team, as well as any welcomed volunteers, will provide their input (through GitHub issues). Eventually, if the project were to be published in a journal (to be determined) where the software ImgBeamer and its accompanying documentation will likely be rigorously reviewed; the expert consultants will review and evaluate the software to establish whether it is fit for public use.

4.4 Verification and Validation Plan Verification Plan

The goal is to uncover any mistakes and reveal any risks (such as misconceptions or coverage gaps) through the supervision and review of the VnV team members. Once most of the work has been done, the work and accompanying documentation shall undergo a vetting process: the VnV team will check whether the documented testing plans and verification process have been accomplished and the requirements fulfilled. The author will then review the documents once more against the VnV checklist [12], before a final review by the rest of the VnV team.

4.5 Implementation Verification Plan

Much of the software will be tested manually by users. This will include checking for any inconsistencies, bugs in the graphical user interface (GUI), and unexpected artifacts in images produced. The image metrics will be tested using unit tests (Section 6). For the all the code implemented, linters will be used as mentioned in Section 4.6. As a control for the image metrics, they will be calculated using the ground truth image as both the input and the reference image (effectively comparing it to itself) to rule out any baseline or unexpected factors in the implementations themselves.

4.6 Automated Testing and Verification Tools

The image quality metric shall be unit tested using pytest for automated testing of any algorithms implemented in Python and QUnit shall be used for those implemented in JavaScript. The unit tests are listed in Section 6. As for linters, flake8 shall be used for Python code and ESLint for JavaScript

code. The sewar python package will be used as a reference implementation in Python for the image quality metrics.

4.7 Software Validation Plan

A *usability* survey will be conducted to evaluate the user experience and whether the GUI is intuitive enough to the intended users as described in the SRS [5]. An *accuracy* survey will be conducted to assess the user-perceived image quality. The trends identified in the surveys results will be compared to the calculated image metrics. As a control for the images produced, a manual test (see T9) will be conducted to verify if an image identical (or with unperceivable difference) to ground truth image can be reproduced. The compared images shall be in the *accuracy* survey for confirmation. This can be compared as well using the image metrics.

5 System Test Description

In this section, the system tests that will be conducted are described in detail. These tests will be used to verify the fulfillment of the requirements as listed in the SRS [5]. Most, if not all, of the tests listed here will be manually performed. Automatic tests will be unit tests as described in Section 6.

5.1 Tests for Functional Requirements

The tests present will verify whether the functional requirements (as listed in the SRS [5]) are met and validate that the outputs are as expected.

5.1.1 Image Import and Export

To satisfy R1 from the SRS [5], any input image of the following formats listed below shall be accepted provided they follow the input data constraints. Some preloaded example images shall be included and be available to the user for use as well. As well as for R6, the software should allow for export of the resulting image.

- PNG
- JPG

- BMP

1. **T1**: Test import for PNG/JPG/BMP format

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The default preloaded image or any non-corrupt (valid) PNG, JPG, or BMP image file of the testers choice.

   Output: The image should be visible and be displayed as expected as the ground truth image. This will be judged and verified by manual visual inspection.

   Test Case Derivation: These are some of the most common image file formats and should be compatible with the software.

   How test will be performed: The user will click the "Load image" button and select an image to import.

2. **T2**: Test PNG Image Export

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: User clicks the *Export* button.

   Output: The output image file should look the same as what is displayed in the "Resulting Image" display. This will be judged and verified by manual visual inspection.

   Test Case Derivation: Not applicable.

   How test will be performed: The user will click the "Export image" button and choose where to save the image file.

5.1.2 Spot Profile and Imaging Parameters

To satisfy R2 and R5 from the SRS [5], the software should accept input from the user to change the width, height, and rotation which, define an ellipsoid shape. This shape is then used as the spot to sample the ground

truth image. To satisfy R3, the software should accept user input for a real positive number for the pixel size. To satisfy R4, the user should be able to specify a subregion (or ROI) for processing.

1. **T3**: Spot Width and Height - Exact-sampling

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Both width and height are set to "100%".

   Output: The spot layout should reflect these changes and display an updated arrangement with the given shape. The resulting image should exhibit the expected **exact-sampling** case as documented by the SRS figures [5]. This will be judged and verified by manual visual inspection.

   Test Case Derivation: Not applicable.

   How test will be performed: Manually, either by using the mouse scroll wheel in the spot content display, or by double-clicking on the width display and entering '100' in prompt that appears.

2. **T4**: Spot Width and Height - Under-sampling

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Both width and height are set to "10%".

   Output: The spot layout should reflect these changes and display an updated arrangement with the given shape. The resulting image should exhibit the expected **under-sampling** case as documented by the SRS figures [5]. This will be judged and verified by manual visual inspection.

   Test Case Derivation: Not applicable.

   How test will be performed: Manually, either by using the mouse scroll wheel in the spot content display, or by double-clicking on the width display and entering '10' in prompt that appears.

3. **T5**: Spot Width and Height - Over-sampling

Control: Manual

Initial State: ImgBeamer loaded and idle.

Input: Both width and height are set to “500%”.

Output: The spot layout should reflect these changes and display an updated arrangement with the given shape. The resulting image should exhibit the expected **over-sampling** case as documented by the SRS figures [5]. This will be judged and verified by manual visual inspection.

Test Case Derivation: Not applicable.

How test will be performed: Manually, either by using the mouse scroll wheel in the spot content display, or by double-clicking on the width display and entering ’500’ in prompt that appears.

4. **T6**: Spot Rotation - Astigmatism

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The angle is set to “45 degrees”, the width is set “60%”, and the height is set to “500%”.

   Output: The spot layout should reflect the updated arrangement with the rotated spots. The resulting image should exhibit the expected **astigmatism** effects as documented in the SRS figures [5]. This will be judged and verified by manual visual inspection.

   Test Case Derivation: Not applicable.

   How test will be performed: How test will be performed: Manually, either by using the mouse scroll wheel in the spot content display, or by double-clicking on the width display and entering ’100’ in prompt that appears. And, by dragging the rotation node in the spot profile display until an angle of 45 is reached.

5. **T7**: Raster Grid / Pixel Size

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The rows and columns are set to “8” for the raster grid.

Output: The resulting image resolution (not image size) should be an “8” by “8” grid image. The “8” rows and “8” columns should be represented in the spot layout display. This will be judged and verified by manual visual inspection.

Test Case Derivation: Not applicable.

How test will be performed: The user shall input the number of rows and columns.

6. **T8**: Subregion / ROI

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The magnification is set to “2x” and the raster grid is set to “8 x 8”.

   Output: Both the spot layout and the resulting subregion displays should have “8 x 8” resolution (rows and columns). The resulting image preview should have “16 x 16” resolution (rows and columns). This will be judged and verified by manual visual inspection.

   Test Case Derivation: Not applicable.

   How test will be performed: The user scroll/zoom in the Subregion (ROI) View display until a magnification of 2x is reached.

7. **T9**: Ground Truth Reproduction

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: An image that is 400x400 pixels is loaded. The rows and columns are set to “200” for the raster grid. The magnification is set to “2x”. Both the spot width and height are set to “100%”.

   Output: The resulting image shall look nearly identical to input image. This will be judged and verified by manual visual inspection.

   Test Case Derivation: Not applicable.

   How test will be performed: The user shall input the number of rows and columns, set the magnification by using the mouse scroll wheel in

the Subregion/ROI display, and set the spot size either by using the mouse scroll wheel in the spot content display, or by double-clicking on the width display and entering '100' in prompt that appears.

5.1.3 Image Quality Metric

To satisfy R7, the user should be able to see a larger number for a resulting image that is closer to ground truth and a smaller number otherwise. The ground truth image used for these tests shall be the default preloaded image or any valid (as per the SRS [5]) image of the tester's choice that is 400 by 400 pixels. Most of these test are conducted manually and judge visually by the tester. As for automated testing, there are control tests for the image metric in section 6.

1. **T10**: High metric value (approximate to exact-sampling)

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The Spot size is set to "120%" with a perfectly circular spot shape. The rows and columns are set to "16". The magnification is set to "2x".

   Output: The score should be greater than "0.8501".

   Test Case Derivation: The resulting image should look somewhat sharp as shown in the SRS figures [5]. This will be judged and verified by manual visual inspection. Preliminary tests were conducted to obtain the score value shown above.

2. **T11**: Low metric value (under-sampling)

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The Spot size is set to "10%" with a perfectly circular spot shape. The rows and columns are set to "16". The magnification is set to "2x".

   Output: The score should be less than "0.8501".

Test Case Derivation: The resulting image should look somewhat overly "sharp" or pixelated as shown in the SRS figures [5]. This will be judged and verified by manual visual inspection. Preliminary tests were conducted to obtain the score value shown above.

3. **T12**: Low metric value (over-sampling)

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The Spot size is set to "500%" with a perfectly circular spot shape. The rows and columns are set to "16". The magnification is set to "2x".

   Output: The score should be less than "0.8501".

   Test Case Derivation: The resulting image should look somewhat "blurry" or "defocused" as shown in the SRS figures [5]. This will be judged and verified by manual visual inspection. Preliminary tests were conducted to obtain the score value shown above.

4. **T13**: Control metric value

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Spot size of a 100%, the same pixel size (image resolution) as the ground truth image, and perfectly circular spot shape. The rows and columns are set to "200". The magnification is set to "2x".

   Output: A number that is near the best or maximum quality metric value (as specified in the SRS [5]). The resulting image should nearly identical to the ground truth image. This will be judged and verified by manual visual inspection. However, The score should be "1.00" (meaning a perfect score), but to allow for a small error margin, it should be greater than "0.9800".

   Test Case Derivation: The resulting image should be identical or have unperceivable differences to the naked eye. This will be judged and verified by manual visual inspection. Preliminary tests were conducted to obtain the score value shown above.

5.2 Tests for Nonfunctional Requirements

The following tests will check if the nonfunctional requirements, as defined in the SRS [5], are met. The emphasis is on the *usability* (NFR2) of the software. In the case of ImgBeamer, simplicity is more important than the actual complexity or correctness. To satisfy the *accuracy* (NFR1) requirement, an image quality and clarity survey shall be conducted to establish what is perceived by the user and the image metric of what a "better" image is. To satisfy the *maintainability* (NFR3) requirement, the code should follow a consistent style and reasonably pass static code analysis as described below. As for *portability* (NFR4), the user should be run the software on their platform and environment of choice.

5.2.1 Usability Testing

1. **T14**: Usability Survey

   Type: Manual

   Condition: A group of intended users, the SRS [5], and the survey questions on topics related to *usability*.

   Result: The completed user surveys (see 8.1).

   Note: The information collected will be aggregated and analyzed for any patterns and changes that may be incorporated to improve the software.

   How test will be performed: The users will be given a series of questions to evaluate the ease-of-use, whether the GUI is intuitive or confusing, etc.

5.2.2 Accuracy Testing

1. **T15**: Image Metric Survey

   Type: Manual

   Condition: A group of intended users, the SRS [5], a list of images of varying quality, and the survey questions on topics related to the image quality and clarity.

Result: The completed user surveys (see 8.2).

Note: The information collected from the surveys will be aggregated and analyzed for any patterns to change or adapt the image metric used according to the rankings.

How test will be performed: The users will be given a series of questions to evaluate a given list of images, rank them based on detail and information content, etc.

5.2.3 Maintainability Testing

1. **T16**: Static code analysis

   Type: Manual

   Condition: The software code will be tested through the use **pylint** for any Python code used and **ESLint** for any JavaScript code used.

   Result: The code linters should list zero warnings (using a specified ruleset, see 8.4), and zero errors.

   How test will be performed: Execute the **pylint** tool on all written Python code and the **ESLint** tool on all written JavaScript code.

2. **T17**: Code Review

   Type: Manual

   Condition: The software source code and the code review checklist (see 8.3)

   Result: The code should respect the defined code review checklist.

   How test will be performed: The author with walkthrough the code manually and check for qualities such as consistent styling, loose variables, variable scoping, unreachable code, sufficient commenting, etc.

5.2.4 Portability Testing

1. **T18**: Various Platforms and Environments

   Type: Manual

Condition: Given any of the supported platforms, the user should be able to set up and run the software with relative ease.

Result: The software runs on the platform of choice with little to no setup including installation, meaning less than `SETUP_MAX_CLICKS` clicks and under `SETUP_MAX_TIME` setup time (see symbolic constants in section 8.5).

How test will be performed: The user will download and run the software on their platform of choice.

6 Unit Test Description

This section focuses on automated testing for individual modules as defined by the MG [2] and MIS [3]. Instructions on how to run the tests and what each folder represents are provided in the project's README file in the source code repository. The unit tests and test image files are located here:
https://github.com/joedf/ImgBeamer/tree/cas741/tests/image_metrics

6.1 Unit Testing Scope

Currently, there's only automated testing of the image metrics calculation (M21). Much of the testing planned is manually due to the complex and subjective nature of image quality. Most of the drawing functionality is handled by the Konva javascript library [8] and thus will not be verified. Although it is possible to test GUIs and visual fidelity to some degree, this is not planned here to due to limited resources (mainly time).

A suite of images (2) is used for testing is provided (with the source code).

No.	File Name	Code	Description
		Ground Truth Images	
1	`original_500-crop.png`	`og`	a virtual image representing metallic grains.
2	`og-fant.png`	`fant`	the original image resampled using Fant's algorithm [7] as 50x50 pixel image and scaled back up using nearest neighbor.
		Control Images	
3	`black.png`	`blak`	all black (`0x00` or `0`) pixels.
4	`white.png`	`wite`	all white (`0xFF` or `255`) pixels.
5	`half.png`	`half`	half white, half black pixels.
6	`gray.png`	`gray`	all pixels are gray (`0x80` or `128`, half of the "colour range" of the 8 bit depth).
7	`og-minor_defects.png`	`mDef`	the original image manually edited to have a few pixels set to white to simulate minor defects.
		Test Images	
8	`c10-010-010.png`	`c010`	10x pixel size, sampled with a spot of 10% by 10%
9	`c10-060-060.png`	`c060`	10x pixel size, sampled with a spot of 60% by 60%
10	`c10-100-100.png`	`c100`	10x pixel size, sampled with a spot of 100% by 100%
11	`c10-120-120.png`	`c120`	10x pixel size, sampled with a spot of 120% by 120%
12	`c10-130-130.png`	`c130`	10x pixel size, sampled with a spot of 130% by 130%
13	`c10-500-500.png`	`c500`	10x pixel size, sampled with a spot of 500% by 500%
14	`c10-060-500.png`	`c6a5`	10x pixel size, sampled with a spot of 60 and 500% at 45°

Table 2: The suite of test images used (all the same size, 500 x 500 px).

6.2 Tests for Functional Requirements

See unit testing scope (6.1) for more information. ImgBeamer's image metric is the only module that will be unit tested to satisfy R7. The metric is tested against the control images (`black.png`, `white.png`, and `gray.png`) from the suite of test images, to assess whether the metric is working as anticipated. A table of the results shall be presented in the VnV report [6].

6.2.1 Module: Image Metrics Calculation (M21)

- **U1**: control - ground truth vs. ground truth

  Type: Automatic

  Input: `original_500-crop.png` and `original_500-crop.png` (see Table 2)

  Output: `1.00` ± ERR_MARGIN

  Test Case Derivation: Should be the highest score/match possible since they are the same image / identical.

  How test will be performed: Running the javascript unit tests.

- **U2**: control - ground truth vs. minor defects

  Type: Automatic

  Input: `original_500-crop.png` and `og-minor_defects.png` (see Table 2)

  Output: `0.99` ± ERR_MARGIN

  Test Case Derivation: Should be very close to the highest score/match possible since they are nearly identical.

  How test will be performed: Running the javascript unit tests.

- **U3**: control - all black vs. all white

  Type: Automatic

  Input: `black.png` and `white.png` (see Table 2)

  Output: `0.00` ± ERR_MARGIN

  Test Case Derivation: Should be the lowest score/match possible since they are complete opposites.

  How test will be performed: Running the javascript unit tests.

- **U4**: control - all black vs. half black and white

  Type: Automatic

  Input: `black.png` and `half.png` (see Table 2)

  Output: `0.29` ± ERR_MARGIN

  Test Case Derivation: Should be $1 - \frac{1}{255}\sqrt{\text{MSE}}$ (following the formula for iNRMSE) as defined in the SRS [5]), where the Mean Square Error (or difference) should be $\frac{1}{2} \cdot (255 - 0)^2 = 32512.5$ (i.e. half "different" meaning half the maximum).

  How test will be performed: Running the javascript unit tests.

- **U5**: control - all white vs. half black and white

  Type: Automatic

  Input: `white.png` and `half.png` (see Table 2)

  Output: `0.29` ± ERR_MARGIN

  Test Case Derivation: Same justification as U4.

  How test will be performed: Running the javascript unit tests.

- **U6**: control - all black vs. all gray

  Type: Automatic

  Input: `black.png` and `gray.png` (see Table 2)

  Output: `0.50` ± ERR_MARGIN

  Test Case Derivation: Should be $1 - \frac{1}{255}\sqrt{\text{MSE}}$ (following the formula for iNRMSE) as defined in the SRS [5]), where the Mean Square Error (or difference) should be $(0 - 128)^2 = 16384$.

  How test will be performed: Running the javascript unit tests.

- **U7**: control - all white vs. all gray

  Type: Automatic

  Input: `white.png` and `gray.png` (see Table 2)

  Output: `0.50` ± ERR_MARGIN

  Test Case Derivation: Same justification as U6, with the slight difference that the MSE is $(255 - 128)^2 = 16129$.

  How test will be performed: Running the javascript unit tests.

- **U8**: control - half black and white vs. all gray

  Type: Automatic

  Input: `half.png` and `gray.png` (see Table 2)

  Output: `0.50` $\pm$ ERR_MARGIN

  Test Case Derivation: One half of the pixels follow the justification of U6, and the other half follow the justification of U7.

  How test will be performed: Running the javascript unit tests.

6.3 Tests for Nonfunctional Requirements

See unit testing scope (6.1) for more information. The image metrics is the only module that will be unit tested to satisfy NFR1.

The image metric needs to be robust and accurate enough for our usecase. The sewar (**L1**) python package (which implements MSE, RMSE, PSNR, UIQ, SSIM, MS-SSIM, SCC, VIF, and more) and following javascript libraries are used for "oracle" comparisons to ImgBeamer's metric (**L2**):

- ssim.web-3.5.0.js (**L3**) – implements SSIM
- image-mse.js (**L4**) – implements MSE, PNSR
- image-ssim.js (**L5**) – implements SSIM
- image-ms-ssim.js (**L6**) – implements MS-SSIM and SSIM

Various libraries with different algorithms/metrics are used here to see how each performs in terms of accuracy using the suite of test images. The expected output values presented here are qualitative as the specific values of each metric for following tests are unknown and may vary widely based on the given images (except for the control images).

A table of the results shall be presented in the VnV report [6].

6.3.1 Module: Image Metrics Calculation (M21)

- **U9**: Image quality ranking

  Type: Automatic

  Input/Condition: The ground truth images (`og` and `fant`) compared to each of the resampled images of varying quality (the "test images" listed in Table 2).

  Output/Result: The expected order (from low to high) of the metric scores is `c500` $\leq$ `c6a5` $\leq$ `c010` $\leq$ `c060` $\leq$ `c100` $\leq$ `c130` $\leq$ `c120`. For the following reasons:

  - `c500` grossly oversamples in all directions
  - `c6a5` is the same, but moderately/reasonably undersamples in one direction

- `c010` grossly undersamples, but conversely the error is localized to each raster cell and not spread out
- `c060` similar, but undersamples to a lesser degree
- `c100` should score reasonably well
- `c130` should score slightly better as there is more coverage, but may spread the error more
- `c120` should score the highest as reasonable compromise between area coverage and error spread

Caveats:

1. These scores are dependent on the raster grid (number of cells and size of the cell or "pixels"). For example, if we only had one large pixel/cell, then the whole image is averaged into one value/signal. If we had a 100 million cells/pixels in both width and height on an image that was only 500 x 500 pixels, then the scores would all be the same (or close to) by averaging/sampling all the same values and relative areas. That is, unless one test would sample with a 100 million percent (of the cell size) for the spot size...
2. The `fant` ground truth image is used as the "ideal scenario" for a downsampled image, as the algorithm is "complete in the sense that all the pixels of the input image under the map of the output image fully contribute to the output image." as said by Karl Fant. This image is used to ensure a fairer comparison and that any image resizing artifacts that we have introduced do not significantly affect the order/trend of the scores. In preliminary tests, the trend was observed to be largely preserved, but with the scores being significantly higher overall.
3. Manually downsampling the ground truth image using different algorithms in an image editor, artifacts such "edge halos‘ (for "Bilinear" and "Lanczos") and "edge loss" (for "Nearest Neighbor") were observed. "Bicubic" performed relatively well, but had blurrier edges than using "Fant". These images are provided with the source code, along with instructions (on how they were produced).

How test will be performed: Running the javascript and python tests. Executing this suite of tests will compare the ground truth image to each test image (see Table 2) using all the different metrics (L1 to L6).

- **U10**: Control testing

  Type: Automatic

  Input: The ground truth images (`og` and `fant`) compared to each of the control test images (as listed in Table 2). Except for using all the other metrics (L1 to L6, except for L2), these are the same control tests as described earlier (U1 to U8).

  Output:

  - U1 Test Variant: same output and justification.
  - U2 Test Variant: same output and justification.
  - U3 Test Variant: same output and justification.
  - U4 Test Variant: a lower score than the base variant, since "structural similarity" is taken into account.
  - U5 Test Variant: same as above.
  - U6 Test Variant: same output and justification, since the compared images have the same ("flat") structure.
  - U7 Test Variant: same as above.
  - U8 Test Variant: a lower score than the base variant, since "structural similarity" is taken into account.

  How test will be performed: Running the javascript and python tests. Executing this suite of tests will compare the "control images" to each over other (as described in U1 to U8), but using the different metrics (L1, L3, L4, L5, and L6).

7 Traceability Between Test Cases and Requirements

Traceability matrices are used to simplify the process of identifying what needs to be changed if a component is modified. An "X" is used to indicate links between items in the table. When a component is changed, the elements marked with an "X" might need to be updated as well.

	R1	R2	R3	R4	R5	R6	R7	NFR1	NFR2	NFR3	NFR4
T1	X										
T2						X					
T3		X			X						
T4		X			X						
T5		X			X						
T6		X			X						
T7			X		X						
T8				X	X						
T9	X	X	X			X					
T10							X				
T11							X				
T12							X				
T13							X				
T14									X		
T15								X			
T16										X	
T17										X	
T18											X
U3 to U10							X	X			

Table 3: Traceability Matrix Showing the Connections Between the Tests and Requirements

References

[1] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995. URL https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale.

[2] J. de Fourestier. Module guide for ImgBeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/Design/SoftArchitecture/MG.pdf.

[3] J. de Fourestier. Module interface specification for ImgBeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/Design/SoftDetailedDes/MIS.pdf.

[4] J. de Fourestier. Problem statement and goals: ImgBeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf.

[5] J. de Fourestier. Software requirements specification for ImgBeamer: Scanning electron microscope image formation, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/SRS/SRS.pdf.

[6] J. de Fourestier. Verification and validation report: ImgBeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/VnVReport/VnVReport.pdf.

[7] Karl M. Fant. A Nonaliasing, Real-Time Spatial Transform Technique. *IEEE Computer Graphics and Applications*, 6(1):71–80, January 1986. ISSN 1558-1756. doi: 10.1109/MCG.1986.276613. Conference Name: IEEE Computer Graphics and Applications.

[8] Anton Lavrenov. Konva.js - JavaScript 2d canvas library, December 2021. URL https://konvajs.org/index.html.

[9] Dr. S. Smith. MG checklist, 2022. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/MG-Checklist.pdf.

[10] Dr. S. Smith. MIS checklist, 2022. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/MIS-Checklist.pdf.

[11] Dr. S. Smith. SRS and CA checklist, 2022. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/SRS-Checklist.pdf.

[12] Dr. S. Smith. System verification and validation plan checklist, 2022. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/VnV-Checklist.pdf.

8 Appendix

This section contains the *usability* and image quality (metric *accuracy*) user survey questions, as well as a code review checklist.

8.1 Usability Survey Questions

- What web browser do you use and what version is it?
- What operating system do you use?
- What is your screen resolution?
- Are you using a mouse and keyboard or a touchscreen?
- How much time (in seconds or minutes) did it take you to find the function you were looking for?
- What was the most useful feature for you?
- On a scale from 1 to 10, how easy was it to set up and start the software?
- On a scale from 1 to 10, how easy was it to use the software?
- What was the least enjoyable feature to use?
- Was everything legible and clearly displayed? If not, what was not?
- Would you consider using the software as a learning or teaching tool?
- Do you have any extra comments or thoughts you'd like to share?

Alternatively, the "Quick and Dirty Usability Scale" [1] may be used instead.

8.2 Image Quality Survey Questions

Given multiple images that are numbered and a ground truth image, the user shall answer the following questions?

- Which image looks to be the clearest in your opinion?
- Which image looks to be the highest quality in your opinion?

- Which image provide the most information or detail?
- Which image is the most similar to or the most representative of the ground truth image?
- Which image is the least pleasant in your opinion?
- Rank the images from the lowest quality (least clear) to the highest quality (most clear) in your opinion?
- Rank the images from least to most informative (or has most of the details of the ground truth image)?
- The user is shown the ground truth image and the reproduced control image (generated using the same image resolution and exact-sampling). Do these images look identical to you? If not, on a scale of 1 to 10, how similar are these images?

8.3 Code Review Checklist

- Does the code follow a consistent style?
- Are there any variables that should not be global?
- Are function names not too long (i.e. less than 40 characters)?
- Are most function well named and clear in what they do?
- Are there any long lines of that can be split into multiple lines?
- Are global variables and functions documented?
- Is each function not too long and sufficiently broken down to accomplish a single task?
- Are comments plentiful, but not noisy?
- Is the code sorted into separate files where reasonable?
- Is there any duplicate that code be avoided?
- Is there any use of jargon, domain-specific, or unclear terms?

- Is all of the code reachable?
- Is the control flow convoluted?
- Is there any unnecessarily obscure code? If necessary, is it commented and explained?
- Could programmer other than the author read the code and briefly understand it?
- Is there any leftover commented code that is not useful?

8.4 Code Linting Ruleset

The code linting ruleset should be the default recommended ruleset plus the following:

- No lines longer than 120 characters.
- No function or variable names are at least 2 characters long and at most 40 characters long. Exceptions:
  - `i`, `j`, `k`: common for iterators in loops
  - `x`, `y`: common for positioning and coordinates
  - `w`, `h`: common for width and height values
  - (ESLint only) `e`: *defacto* standard for javascript events
  - `c`, `d`: used heavily in ImgBeamer's image data manipulation. Otherwise, the code gets very difficult to read when dealing multiple arrays of data on a lower level
- No "magic numbers". Exceptions:
  - `0`, `1`: common for array starting indexes
  - `2`: common for getting half of a value
  - `3`, `4`: used heavily for managing RGBA values/data on a lower level
  - `-1`: common value for failure or an error
  - `100`: common for percentages

  - `180`: common for angle conversions
  - `255`: the maximum value for an 8-bit pixel value. ImgBeamer is limited to 8-bit images as specified in the SRS [5] and is unlikely to change as specified in the MG [2].

- Warn about global variables. Each shall be carefully considered whether it should remain a global variable. Library names are exempt.
- (flake8 only): ignore indentation (tabs vs. spaces) style `--ignore=W191`.

8.5 Symbolic Constants

These are *symbolic* values used in the descriptions and text in this document that are subject to change. They listed and defined here to make it easy to change and reduce the possibility of mistakes when updating this document.

- `SETUP_MAX_CLICKS = 4` clicks
- `SETUP_MAX_TIME = 3` minutes
- `ERR_MARGIN = 0.02` (2%)