GSoC 2024 Final Report

Description of the goals of the project

InVesalius is a cross-platform (Windows/ linux/ Apple OS) 3D Medical visualization and neuro-navigation tool, developed since 2001 by Centro de Tecnologia da Informação Renato Archer (CTI), Brazil. The software is mainly used for rapid prototyping, teaching, and forensics in the medical field. (https://invesalius.github.io/about.html).

My goal towards this project was to develop tools to facilitate users to activate the capture of logs and errors. The deliverables were as follows:

1. Graphic interface integrated to InVesalius that allows the user to activate the tool and save the logs, supporting respective log levels (debug, info, warn, error or critical). It should be possible to save the sequence of events in a text file.
2. A separate text window for console stream output.
3. Log support to all InVesalius functions and functionalities.
4. Add provision/tools for error catching.

https://summerofcode.withgoogle.com/programs/2024/projects/zYrNyRVw

What I did

After going through and getting an understanding of the code, I concluded to work towards fulfilling my goal in following steps:

a. Front end user interface for accepting user preferences for logging
b. A separate text window for console stream output
c. Backend tools for setting up logging support as per user preferences,
d. Propagating logging message appropriately as per need, and
e. Tools for error catching support.

To do the development, I created a branch, and added code to do the following:
Github link for the branch: https://github.com/rishabh-ranjan1/invesalius3/tree/addLogging

a. Front end user interface for accepting user preferences for logging: We have finalized the interface for accepting user preferences for logging as shown in Figure 1.
A class was added for the implementation. The logging option are stored in a json file in the configuration folder. If not found, then default options are selected. User have options for console and/or file logging, logging level selections for each, and others needed. Logging file name may be modified using the 'Modify' buttons with the interface shown in Figure-2.

b. Separate text window for console stream output:
Few classes were added to support popping of a separate window for displaying console stream output including logging messages (if chosen by user). This interface is shown in Figure-3.

c. Backend tools for setting up logging support as per user preferences
A class was added to read up the logging configuration json file mentioned above to do and/or modify the setup as per user preferences.

d. Propagating logging message: My mentors approved my proposal to propagate logging messages to different invesalius functions using decorators.

My prime reasons for the use of decorators are as follows:

- The log message code is not repeated, and localised for making global changes.
- There is no risk of introducing error in existing code as existing code is not touched. It was important due to absence of test programs.
- If there is an error or change is needed in logging, it can be rectified in the decorator alone.

e. Tools for error catching support:
Decorators were written for error catching support, for the reasons mentioned above.

The current state

The code developed for logging interface, separate window for console streaming, logging tools, and the decorators were evaluated by the mentors, and were merged upstream. The merge reference is https://github.com/invesalius/invesalius3/pull/780.

Changes in the implemented decorator function and addition of new decorator functions are ongoing as needed. For logging of functions called, I am now passing file name, name of the class (if any) and function name to the decorator as parameters. File name is automatically passed, but I am figuring if class name and function name could be obtained automatically. Another decorator, without any parameter, is provided where function name could be automatically obtained.

What is ongoing/left to do

I have added the decorator to functions in image/bitmap reader functions I felt appropriate. I have also added decorator to the segmentation functions. Addition of decorators to other functions is ongoing.

Any challenges or important things you learned during the project

A decorator is a design pattern tool in Python for wrapping code around functions or classes (defined blocks). This design pattern allows a programmer to add new functionality to existing functions or classes without modifying the existing structure. I am really excited to come across, learn and use this design pattern, and am continuing to explore.

The one challenge I encountered using python language is not figuring if a macro-like structure be crafted. The advantage of the structure is that a copy of the macro is made at the place of call, and thus can obtain the information at the place of invocation.

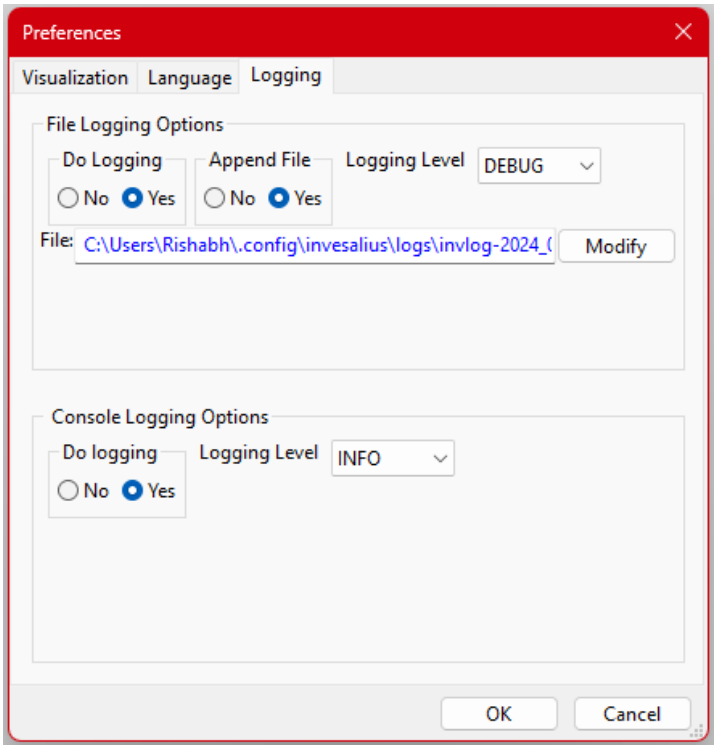

Figure 1: Logging Interface

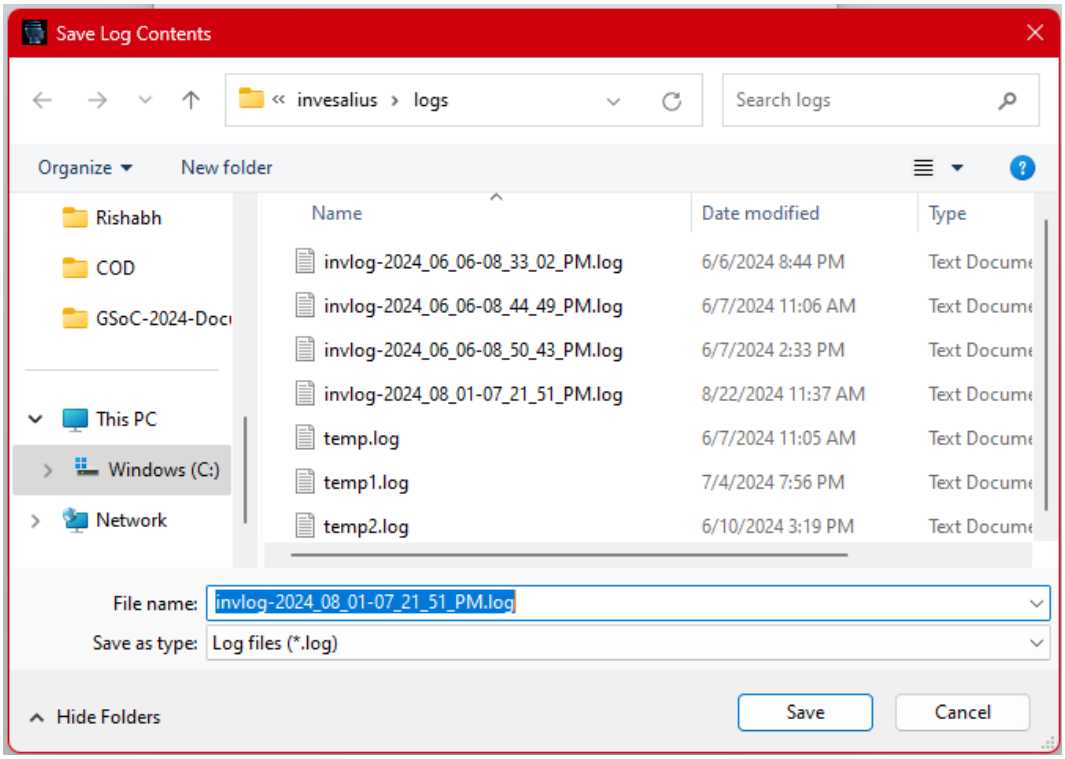

Figure 2: Log file selection

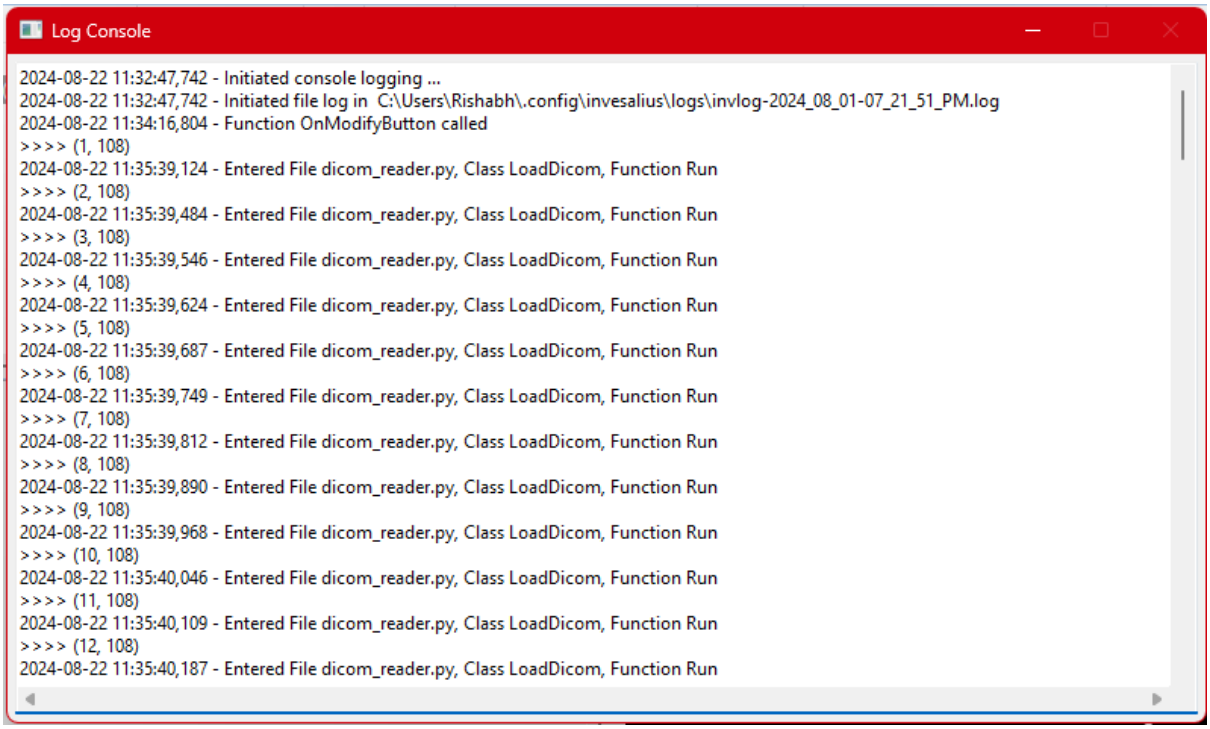

Figure 3: Separate window for console stream output, including log messages (if chosen)