ORNL/TM-2024/3632

Peregrine Software Development: Report on the Code Conversion From Python to C++

Jerome Benoit
Luke Scime
Vincent Paquit

September 2024

M3CT-24OR1305052

ORNL/TM-2024/3632

Advanced Materials and Manufacturing Technologies Program

PEREGRINE SOFTWARE DEVELOPMENT: REPORT ON THE CODE CONVERSION FROM PYTHON TO C++

Jerome Benoit
Luke Scime
Vincent Paquit

September 2024

M3CT-24OR1305052

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

CONTENTS III
LIST OF FIGURES IV
ABBREVIATIONS V
ABSTRACT VI
1. OVERVIEW 1
  1.1 PEREGRINE 1
  1.2 WORK PACKAGE OBJECTIVES 2
2. PEREGRINE USER INTERFACE AND CODE CONVERSION 2
  2.1 COMPARING PEREGRINE'S PYTHON AND C++ ENVIRONMENTS 2
    2.1.1 Python 2
    2.1.2 C++ 2
  2.2 DATA FILE FORMAT 3
    2.2.1 Python 3
    2.2.2 C++ 3
  2.3 PEREGRINE USER INTERFACE 3
  2.4 PEREGRINE DATA LOADER 4
  2.5 PEREGRINE METADATA DIALOG 5
  2.6 PEREGRINE DOCUMENT VIEW 7
    2.6.1 Mainframe Window 9
    2.6.2 Anomalies View 10
    2.6.3 Main graph widget 11
    2.6.4 Details View 12
    2.6.5 Position Window 14
    2.6.6 Layer Details Window 14
    2.6.7 Settings Window 15
  2.7 SOURCE CODE REPOSITORY 16
CONCLUSION 18
APPENDIX A. SYSTEM CONFIGURATION A-1

LIST OF FIGURES

Figure 1. Peregrine user interface displaying (right) the classification results of a single layer of a build, using one color for each type of detectable feature. 1
Figure 2. (Top) Python and (bottom) C++ GUIs for Peregrine. 4
Figure 3. Peregrine data loader windows in (top) Python and (bottom) C++....5
Figure 4. Peregrine metadata dialog boxes in (left) Python and (right) C++....6
Figure 5. Document view in Peregrine for (top) Python and (bottom) C++....7
Figure 6. Additional document views in Peregrine....8
Figure 7. Mainframe window....10
Figure 8. Anomalies view. 11
Figure 9. Details view. 13
Figure 10. Position window. 14
Figure 11. Layer Details window....15
Figure 12. Settings window. 16
Figure 13. MDI architecture....18

ABBREVIATIONS

API	application programming interface
GUI	graphic user interface
JSON	JavaScript Object Notation
MDI	multi-documents interface architecture
ORNL	Oak Ridge National Laboratory
SSD	solid-state drive

ABSTRACT

This work package seeks to convert the Peregrine software tool from its original Python implementation to a production version based on the C++ language.

Peregrine is a powerful research platform with a multitude of advanced data analytics and data visualization functionalities. Developed by scientists to explore multimodal and multidimensional data related to the production of components using powder bed additive manufacturing processes, the tool implements state-of-the-art algorithms to assist machine users in making build or part quality determinations. Given that Peregrine is data-intensive, the goal of this conversion is to enhance the tool's flexibility and interactivity and reduce the number of code dependencies to facilitate its deployment as part of the ongoing technology transfer campaign.

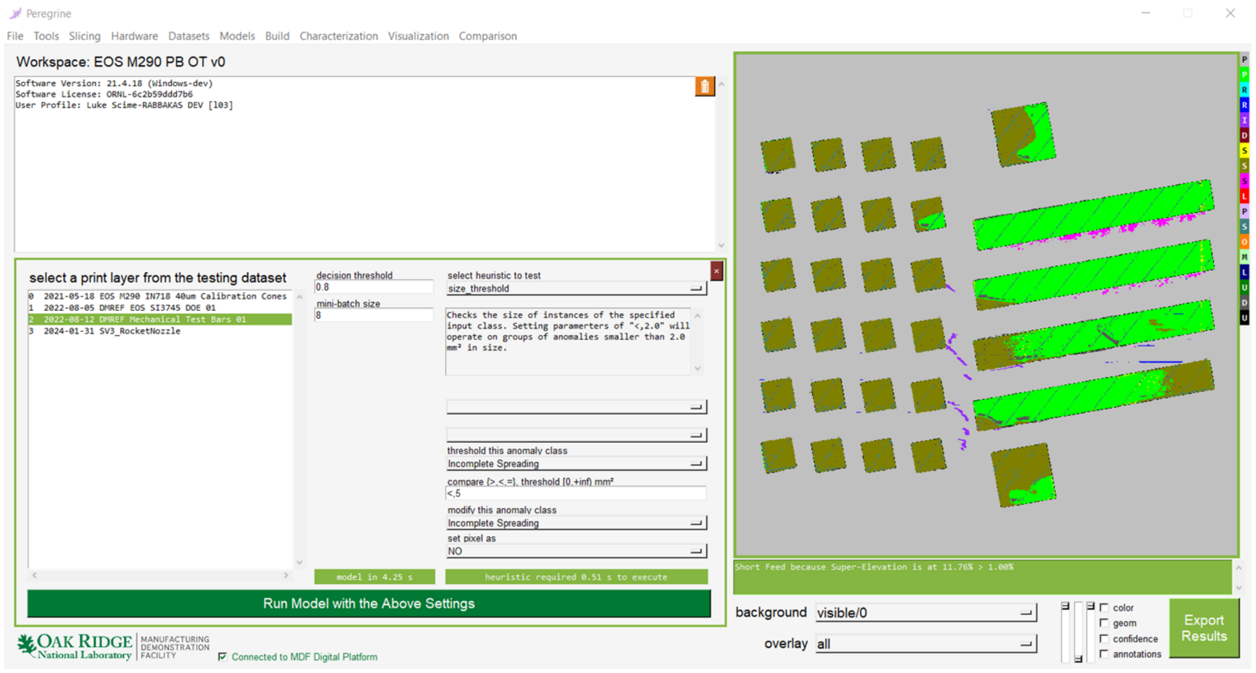
This brief document provides an overview of Peregrine's functionalities and capabilities, along with a detailed description of the core functionalities that have been implemented to date in the new C++ version. This document serves as a development update at the end of the first year of the ongoing conversion and will be regularly updated as progress continues.

1. OVERVIEW

1.1 PEREGRINE

Over the past 5 years, a team of scientists and engineers from the Manufacturing Demonstration Facility located at Oak Ridge National Laboratory (ORNL) has been developing a software tool called Peregrine allowing *"users to collect, analyze, and visualize both in-situ sensor and ex-situ characterization data in order to assist engineers when making build or part quality determinations for powder bed additive manufacturing (AM) processes."*[1]

Peregrine primarily leverages image-based sensor data to assess the quality of the top layer in powder bed systems. Images collected from one or multiple imaging modalities are analyzed using a custom neural network architecture that performs pixel-wise classification of the texture features present in the image data. The primary function of Peregrine is to provide interactive visualization of the in-situ data classification results to assist machine users in their quality control assessment (Figure 1). This capability has been successfully demonstrated on multiple powder bed systems using three technologies, (i.e., electron beam melting, laser powder bed fusion, and binder jetting).


Peregrine
File Tools Slicing Hardware Datasets Models Build Characterization Visualization Comparison
Run Model with the Above Settings
Export Results
color
geom
confidence
annotations
OAK RIDGE National Laboratory
MANUFACTURING DEMONSTRATION FACILITY

Workspace: EOS M290 PB OT v0

```
Software Version: 21.4.18 (Windows-dev)
Software License: ORNL-6c2b59ddd7b6
User Profile: Luke Scime-RABBAKAS DEV [103]
```

select a print layer from the testing dataset

```
0  2021-05-18 EOS M290 IN718 40um Calibration Cones
1  2022-08-05 DMREF EOS SI3745 DOE 01
2  2022-08-12 DMREF Mechanical Test Bars 01
3  2024-01-31 SV3_RocketNozzle
```

decision threshold: 0.8

mini-batch size: 8

select heuristic to test: size_threshold

```
Checks the size of instances of the specified
input class. Setting paramerters of "<,2.0" will
operate on groups of anomalies smaller than 2.0
mm² in size.
```

threshold this anomaly class: Incomplete Spreading

compare {>,<,=}, threshold [0,+inf) mm²: <,5

modify this anomaly class: Incomplete Spreading

set pixel as: NO

```
model in 4.25 s
heuristic required 0.51 s to execute
Short Feed because Super-Elevation is at 11.76% > 1.00%
```

background: visible/0

overlay: all

☑ Connected to MDF Digital Platform

Figure 1. Peregrine user interface displaying (right) the classification results of a single layer of a build, using one color for each type of detectable feature.

[1] Quoted from the Peregrine user manual

1.2 WORK PACKAGE OBJECTIVES

Since its initial implementation, Peregrine has evolved to integrate additional data modalities and now supports more complex data processing tasks, such as multiclass contextual segmentation, materials property predictions, and real-time modeling and simulation. As a result, the amount of data handled by Peregrine represents tens of gigabytes and necessitates the development of computationally efficient algorithms and visualization widgets for 2D and 3D rendering. In this domain, Python has known limitations. It also requires a lengthy list of dependencies to achieve these objectives. The goal in this work package is to explore the possibility of converting Peregrine from Python to C++ to take advantage of the computing efficiency and flexibility it offers and consolidate the number of libraries and dependencies involved. The objective for this year was to recreate Peregrine's user interface using more flexible data rendering techniques, and to find an alternative to the Python data format to save files.

2. PEREGRINE USER INTERFACE AND CODE CONVERSION

2.1 COMPARING PEREGRINE'S PYTHON AND C++ ENVIRONMENTS

2.1.1 Python

The current version of Peregrine is version 21.6. It uses the 3.9 version and 64 bits; of Python and requires 59 dependencies. Several of these dependencies are only needed for basic functionalities, and others are core parts of the software tool:

- **Module Tkinter** is the user interface backbone library that was used to create Peregrine's graphical user interface (GUI), including some of the 2D visualization widgets.

- **Module Cv2** refers to the OpenCV library that provides numerous algorithms for image processing and computer vision. It includes functionalities to access cameras connected to a computer, to simplify camera calibration, and to process images using classic image processing algorithms.

- **Module Json** is a library to manipulate JavaScript Object Notation (JSON) data.

- **Module Matplotlib** is a library to create static and interactive 2D data visualizations.

- **Module Open3D** is a library to create interactive 3D data visualizations.

- **Module Numpy** is a library for numerical calculation that uses a specific data representation to accelerate computation. The library also allows users to save data in its proprietary format (i.e., npy and npz) that is commonly used to store neural network models.

2.1.2 C++

The C++ version of Peregrine is currently being developed in a Windows environment using Microsoft Visual Studio 19 IDE. However, the code is being written to be portable without using any Windows-specific application programming interface (API). Additionally, this project has selected cross-platform libraries that will allow developers to compile the same code on Linux and macOS systems.

Also, for the GUI, this work is using Qt, which is portable to most platforms in use today, including Linux and macOS.

- **Compiler** is ISO C++ 17 // Core C++ toolkit, 64 bits.

- **Qt library** is a popular cross-platform C++ library for advanced GUI design with multiple submodules offering equivalent capabilities to most of the Python modules used by Peregrine.

- **HDF5 library** is an open-source file format that supports large, complex, heterogeneous data.

- **OpenCV library** is identical to its Python counterpart.

- **OpenMP library** allows easy multithreading.

2.2 DATA FILE FORMAT

2.2.1 Python

Most data files are saved in Numpy's npy and npz formats. Although pure npy and npz files can be easily read from C++, some Peregrine files are mixed with data in Python's "Pickle" format. The Pickle format includes data type information and relies on Python to recreate the original data structure. This process is too complex to implement in the C++ application because it would require adding a significant portion of the Python codebase to the data reader.

Given the difficulty of accessing "pickled" data from the C++ application, this project opted to use a Python script to first convert the data into a format that C++ can easily read. HDF5 was chosen because it offers APIs for both Python and C++ and it is a binary format. As a binary format, HDF5 is more compact and faster to read than nonbinary formats such as JSON. These characteristics are important because of the large volumes of data that are handled, where file size is a significant factor. Additionally, HDF5 is portable across multiple platforms, including Linux and macOS, unlike the Microsoft OLE compound document format, which is limited to Windows.

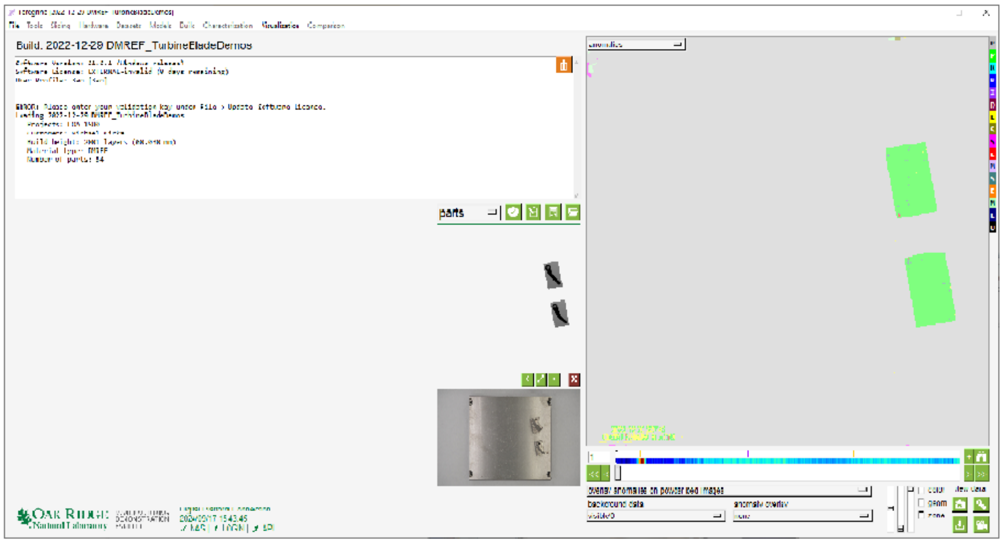
2.2.2 C++

As mentioned previously, most files are handled using HDF5. This project developed a generic Python script to translate Numpy npy/npz files into HDF5. In this context, *generic* means that the script does not require specific knowledge of the source files to perform the translation. If the source files change or new files are introduced, the script will still function correctly and convert the source files to HDF5 without modifications. This project has also developed a Python script to translate HDF5 files back into npy or npz formats.

2.3 PEREGRINE USER INTERFACE

The C++ version of the GUI is based on a multi-documents interface architecture (MDI). Notably, the Python version can also display multiple windows and can display all the views displayed by the current C++ version and much more. However, the Python version main window does not offer the same level of flexibility provided by the C++ version. In the C++ version, all windows can be resized and repositioned freely by the user. Figure 2 shows an example of each GUI version in Peregrine.

Python Version


File Tools Slicing Hardware Datasets Models Builds Characterization Visualization Comparison
parts
OAK RIDGE National Laboratory
MANUFACTURING DEMONSTRATION FACILITY

Build. 2022-12-28 DMREF_TurbineBladeDemos

```
Software Version: 21.2.1 (Windows release)
Software License: EXTERNAL-invalid (0 days remaining)
User Profile: [illegible]

ERROR: Please enter your validation key under File > Update Software License.
Loading 2022-12-28 DMREF_TurbineBladeDemos
   Project: [illegible]
   Customer: [illegible]
   Build height: 2001 layers ([illegible] mm)
   Material type: DMREF
   Number of parts: 54
```

C++ Version

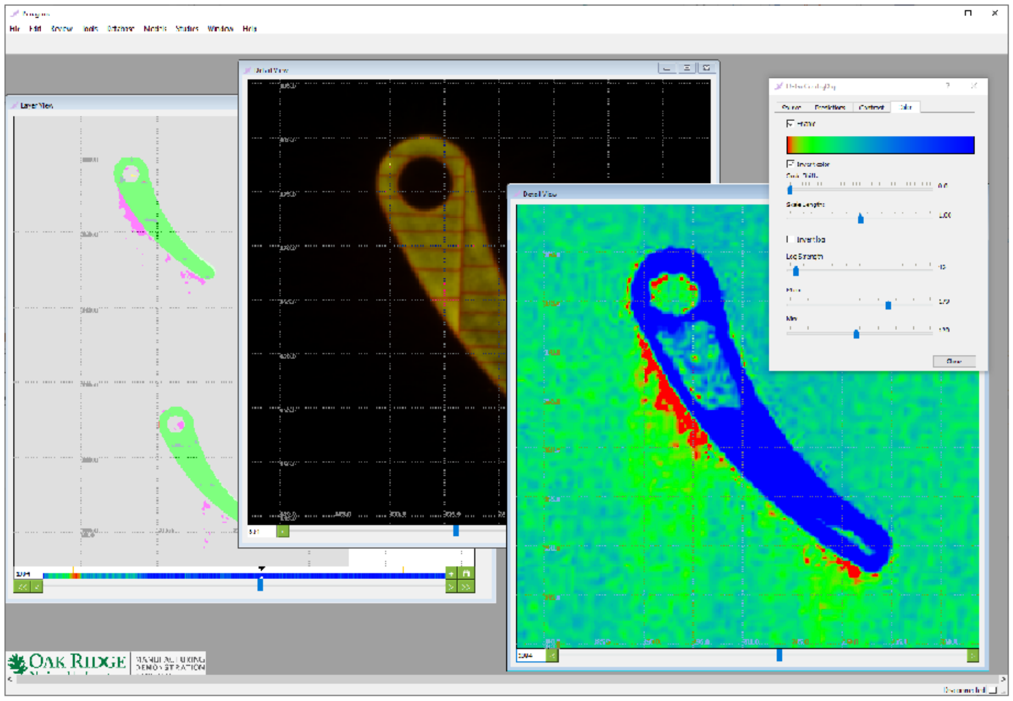
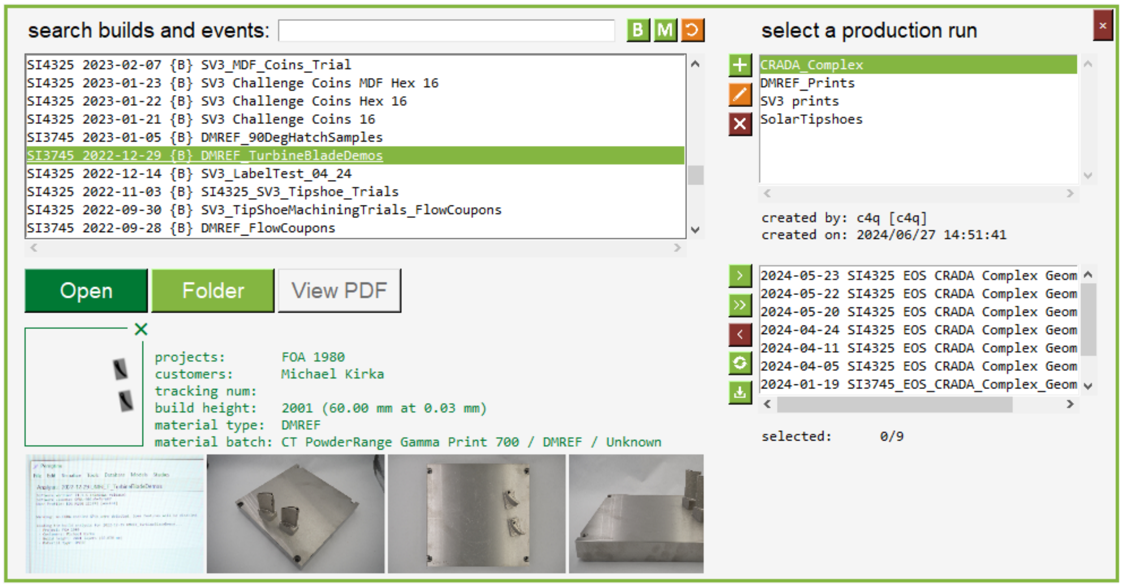

File Edit Screen Tools Datasets Models Studies Window Help
Layer View
Detail View
OAK RIDGE
MANUFACTURING DEMONSTRATION

Figure 2. (Top) Python and (bottom) C++ GUIs for Peregrine.

2.4 PEREGRINE DATA LOADER

The Peregrine Data Loader provides functionalities to select and previsualize available datasets. This project replicated most of the loading functionalities, but notably, the Python version was changed last spring. The C++ version will to be reworked to match the new version. Both versions as they currently exist are shown in Figure 3.

Python Version



```
SI4325 2023-02-07 {B} SV3_MDF_Coins_Trial
SI4325 2023-01-23 {B} SV3 Challenge Coins MDF Hex 16
SI4325 2023-01-22 {B} SV3 Challenge Coins Hex 16
SI4325 2023-01-21 {B} SV3 Challenge Coins 16
SI3745 2023-01-05 {B} DMREF_90DegHatchSamples
SI3745 2022-12-29 {B} DMREF_TurbineBladeDemos
SI4325 2022-12-14 {B} SV3_LabelTest_04_24
SI4325 2022-11-03 {B} SI4325_SV3_Tipshoe_Trials
SI4325 2022-09-30 {B} SV3_TipShoeMachiningTrials_FlowCoupons
SI3745 2022-09-28 {B} DMREF_FlowCoupons
```

```
CRADA_Complex
DMREF_Prints
SV3 prints
SolarTipshoes
```

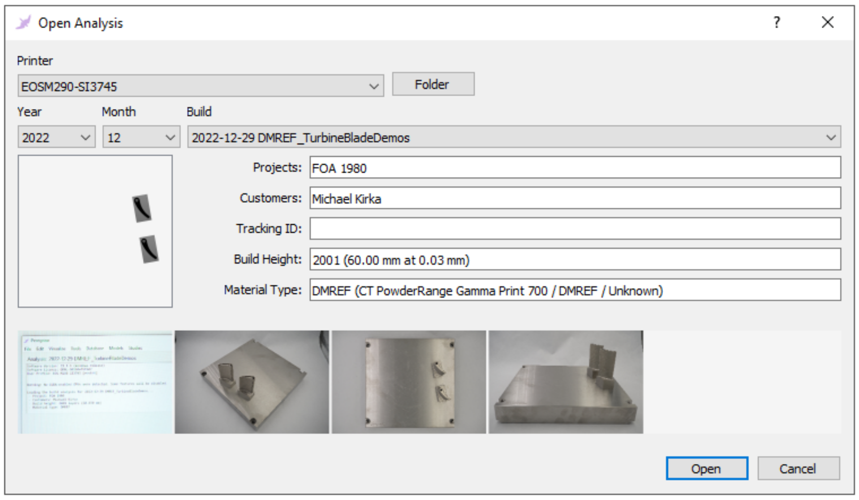
```
created by: c4q [c4q]
created on: 2024/06/27 14:51:41
```

```
projects:       FOA 1980
customers:      Michael Kirka
tracking num:
build height:   2001 (60.00 mm at 0.03 mm)
material type:  DMREF
material batch: CT PowderRange Gamma Print 700 / DMREF / Unknown
```

```
2024-05-23 SI4325 EOS CRADA Complex Geom
2024-05-22 SI4325 EOS CRADA Complex Geom
2024-05-20 SI4325 EOS CRADA Complex Geom
2024-04-24 SI4325 EOS CRADA Complex Geom
2024-04-11 SI4325 EOS CRADA Complex Geom
2024-04-05 SI4325 EOS CRADA Complex Geom
2024-01-19 SI3745_EOS_CRADA_Complex_Geom
```

```
selected:      0/9
```

C++ Version



Printer	EOSM290-SI3745
Year	2022
Month	12
Build	2022-12-29 DMREF_TurbineBladeDemos
Projects:	FOA 1980
Customers:	Michael Kirka
Tracking ID:	
Build Height:	2001 (60.00 mm at 0.03 mm)
Material Type:	DMREF (CT PowderRange Gamma Print 700 / DMREF / Unknown)

Figure 3. Peregrine data loader windows in (top) Python and (bottom) C++.

The loader interface consists of:

- a list view for the newest Python version now replacing a drop-down button (the drop-down button has been implemented, but this project is planning to make this change, as well),
- metadata viewers, and
- image viewers to quickly identify known parts based on their geometries.

2.5 PEREGRINE METADATA DIALOG

Figure 4 provides examples of the Peregrine metadata dialog boxes. The Python version on the left is complex and difficult to extend, and the C++ version on the right is more streamlined and easier to extend. Multiple edit boxes have been replaced with a single tree control, which simplifies adding or deleting metadata entries. By double-clicking an entry, a dialog box opens, allowing users to modify the value of the selected entry.

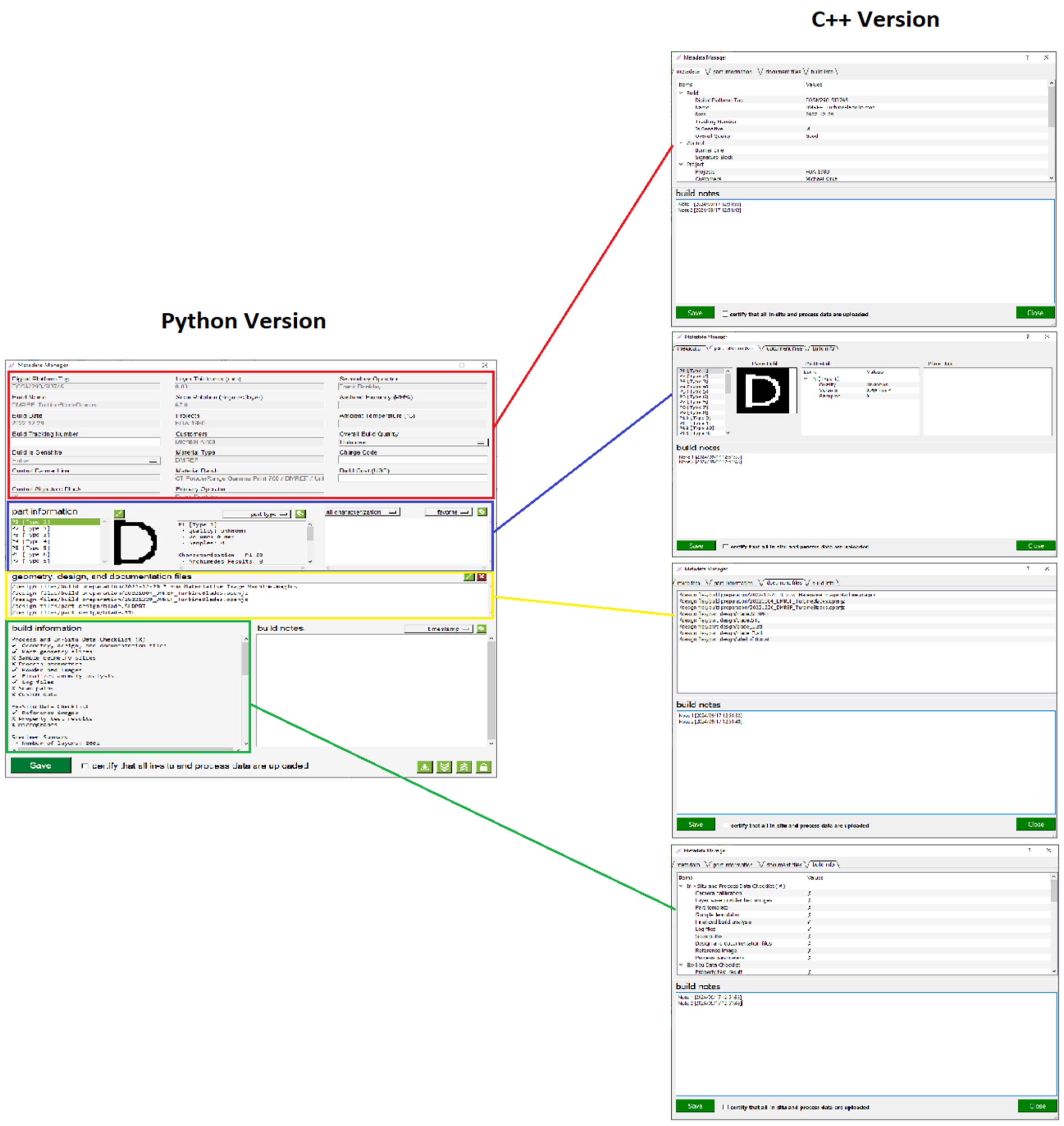

C++ Version
Python Version
Metadata Manager
part information
geometry, design, and documentation files
build information
build notes
Save
I certify that all in-situ and process data are uploaded
Close

Figure 4. Peregrine metadata dialog boxes in (left) Python and (right) C++.

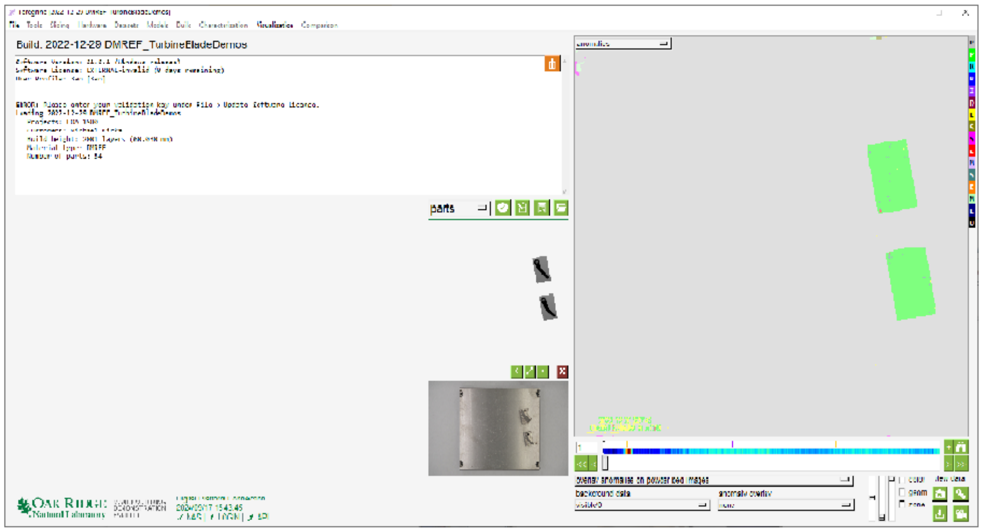
Graphic elements are organized into panes within a tab control, with only one pane visible at a time. This design prevents overwhelming the user with too many controls at once. When additional items need to be displayed, more panes can easily be added.

Various functionalities are accessed through multiple context menus, each adapted to the specific control being clicked.

2.6 PEREGRINE DOCUMENT VIEW

As previously shown, the C++ version's GUI is based on an MDI architecture. This type of interface gives users flexibility in arranging windows or views. A *view* represents a visualization of some of the data contained in a document and can be graphical, textual, dialog-based, and more. An example is shown in
Figure 5. In this specific case, the architecture follows a single-document model with multiple views.

Python Version


File Tools Slicing Hardware Datasets Models Build Characterization Visualization Comparison
parts
OAK RIDGE National Laboratory

Build. 2022-12-28 DMREF_TurbineBladeDemos

```
Software Version: 21.2.1 (Windows release)
Software License: EXTERNAL-invalid (0 days remaining)
User Profile: [illegible]


ERROR: Please enter your validation key under File > Update Software License.
Loading 2022-12-28 DMREF_TurbineBladeDemos
   Projects: [illegible]
   [illegible]
   Build height: 2001 layers ([illegible] mm)
   Material type: DMREF
   Number of parts: 54
```

C++ Version

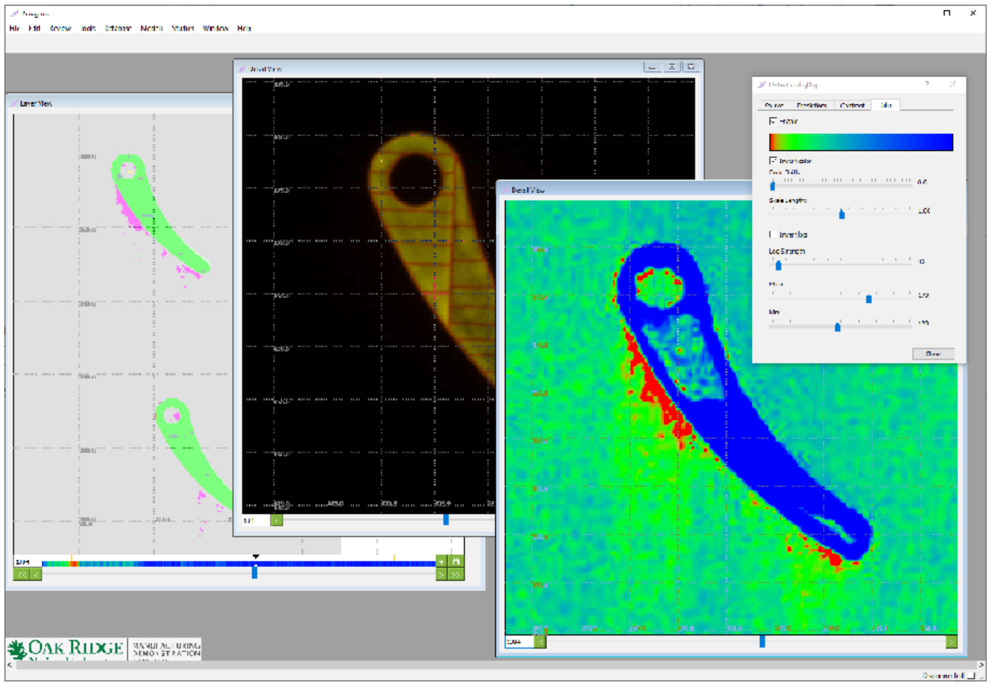


Figure 5. Document view in Peregrine for (top) Python and (bottom) C++.

In Figure 6, more views have been added and displayed in a tile formation.

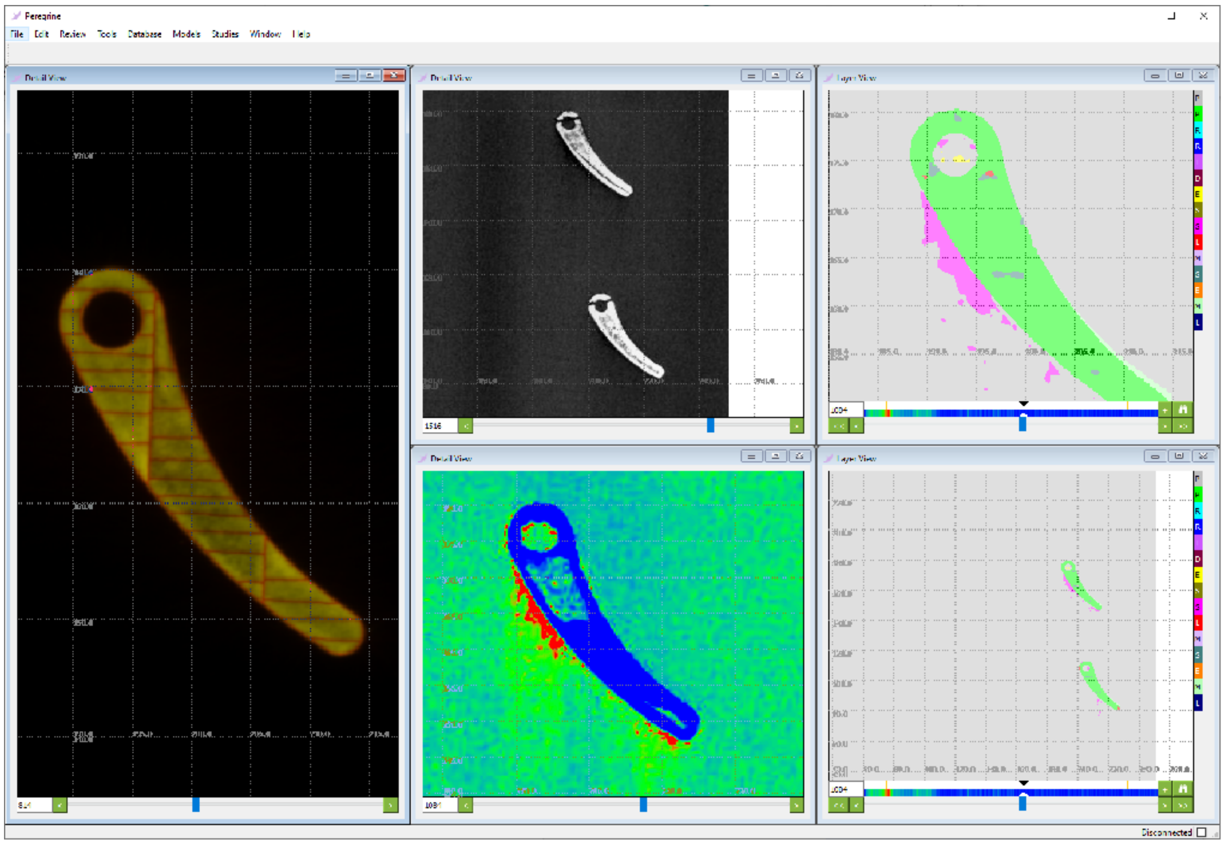


Figure 6. Additional document views in Peregrine.

The interface provides standard "tile" and "cascade" options, and the application can be resized freely.

The application consists of a mainframe, in which views are displayed, and a document. The *document* is a logical entity that handles all data transactions, such as with npy, npz, and others.

Currently, two types of views are available: the "anomalies view" and the "details view." Additional view types are under development. Multiple instances of both types of views can be displayed simultaneously; with no preset limit on the number of views, which is only constrained by available memory and processor speed.

Views are child windows of the mainframe, meaning they are displayed on top of the mainframe and interact with both the mainframe and the document. Views also have satellite windows that provide additional information, such as positions, color settings, and layer details.

Because of the importance of layer numbers in 3D printing, each view includes its own layer selector. Views can be synchronized by layer, so changing the layer in one view will update all linked views to the same layer. Similarly, positions can be synchronized, so panning or zooming in one view adjusts the position in other linked views.

Views share many common functionalities:

- Title bar
- Resizing
- Panning
- Zooming
- Context menus
- Layer synchronization
- Position synchronization
- Satellite windows
- Layer selector
- Clipboard support

2.6.1 Mainframe Window

The mainframe window (Figure 7) hosts all the views. It consists of the following elements

- Title bar
- Menu bar
- Tools bar
- Client area
- Status bar

The views are hosted in the client area of the mainframe, which is the large dark gray area in the middle of Figure 7.

Mainframe window

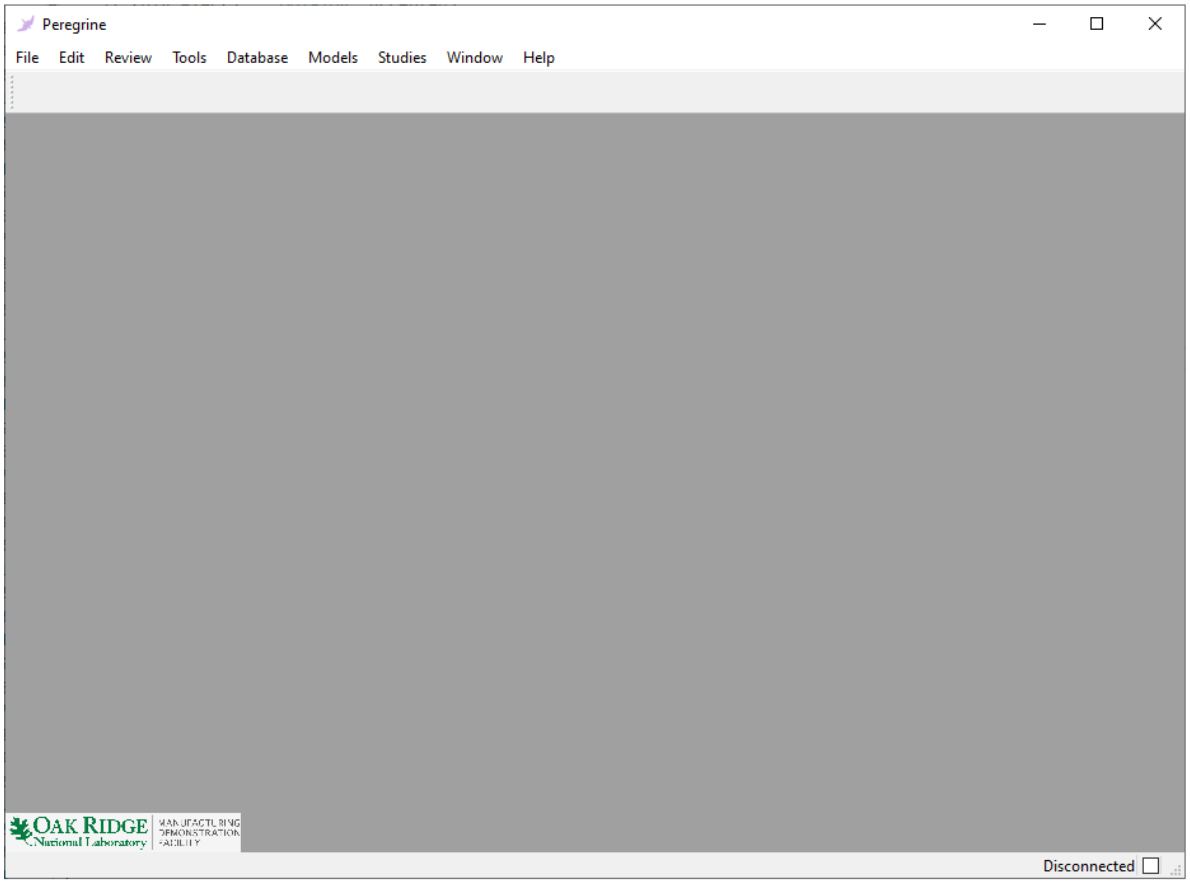


Figure 7. Mainframe window.

2.6.2 Anomalies View

The Anomalies view (Figure 8) displays the print anomalies in a similar manner as the Python version. It provides the following features:

- Main graph displays the actual data
- Color legend describes the color assignments
- Similarity graph shows how the current layer compares with other layers
- Layer selector allows the user to select the layer

This view can also display two satellite windows: the Position window and the Layer Details window. These satellite windows are child windows of the view and can be positioned anywhere on the screen, including outside the application's mainframe.

Anomalies View

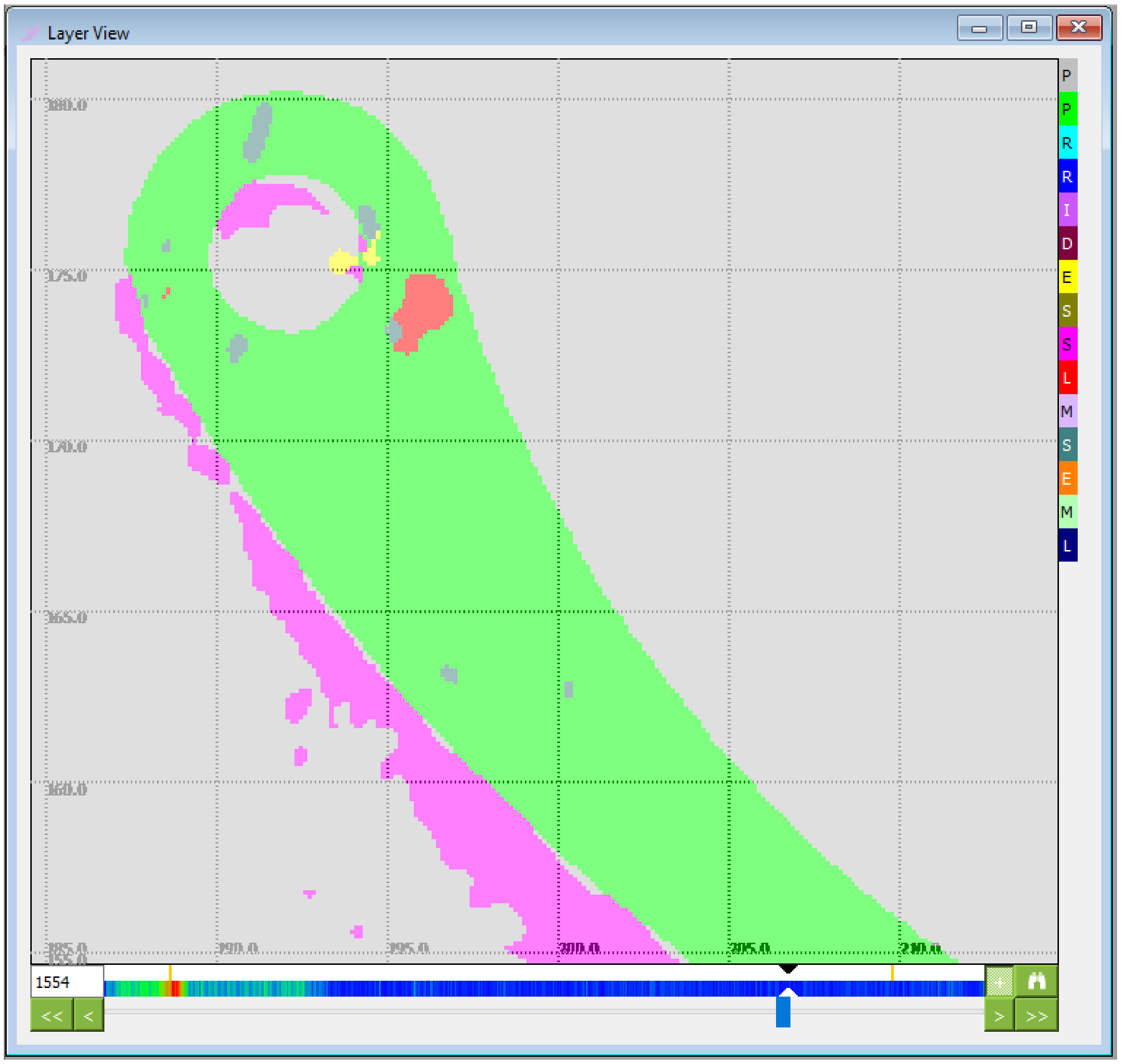


Figure 8. Anomalies view.

2.6.3 Main graph widget

The main graph is displayed using a custom Qt widget that was developed in this work. In Qt, a widget is an encapsulated graphic object that can be easily reused and extended. This widget takes an OpenCV raster and displays it. The base version of the widget provides the following features:

- Color display
- Zooming
- Panning
- Position synchronization
- Layer synchronization

Designed for speed, the widget can handle a raster of any size. It uses multithreading to enhance rendering performance, and the rendering algorithm in this work has a size complexity of $O(1)$ instead of $O(n^2)$. This complexity means that the rendering speed is independent of raster size, allowing a large 100,000 × 100,000 pixel raster to display, zoom, and pan as responsively as a smaller 1,000 × 1,000 pixel raster.

There is no preset limit on the size other than the amount of memory available. A single 100,000 × 100,000 pixels, one-channel raster requires just over 40 Gb of RAM.

The widget comes in three types:

- Single channel without grid
- Single channel with grid
- Multi-channels with grid

The Anomalies view uses the single channel with grid version. The grid can be turned on and off. It provides position guidelines to the user. The spacing and number of lines adjust automatically when panning, zooming, or resizing the view. The numbers shown are in physical units derived from the metadata.

This custom widget significantly contributes to maintaining the application's responsiveness.

2.6.4 Details View

The "Details view" can display various type of raster:

- single channel
- plain colors
- composite colors
- multi channels

It can also overlay the print anomalies on these images.

Details View

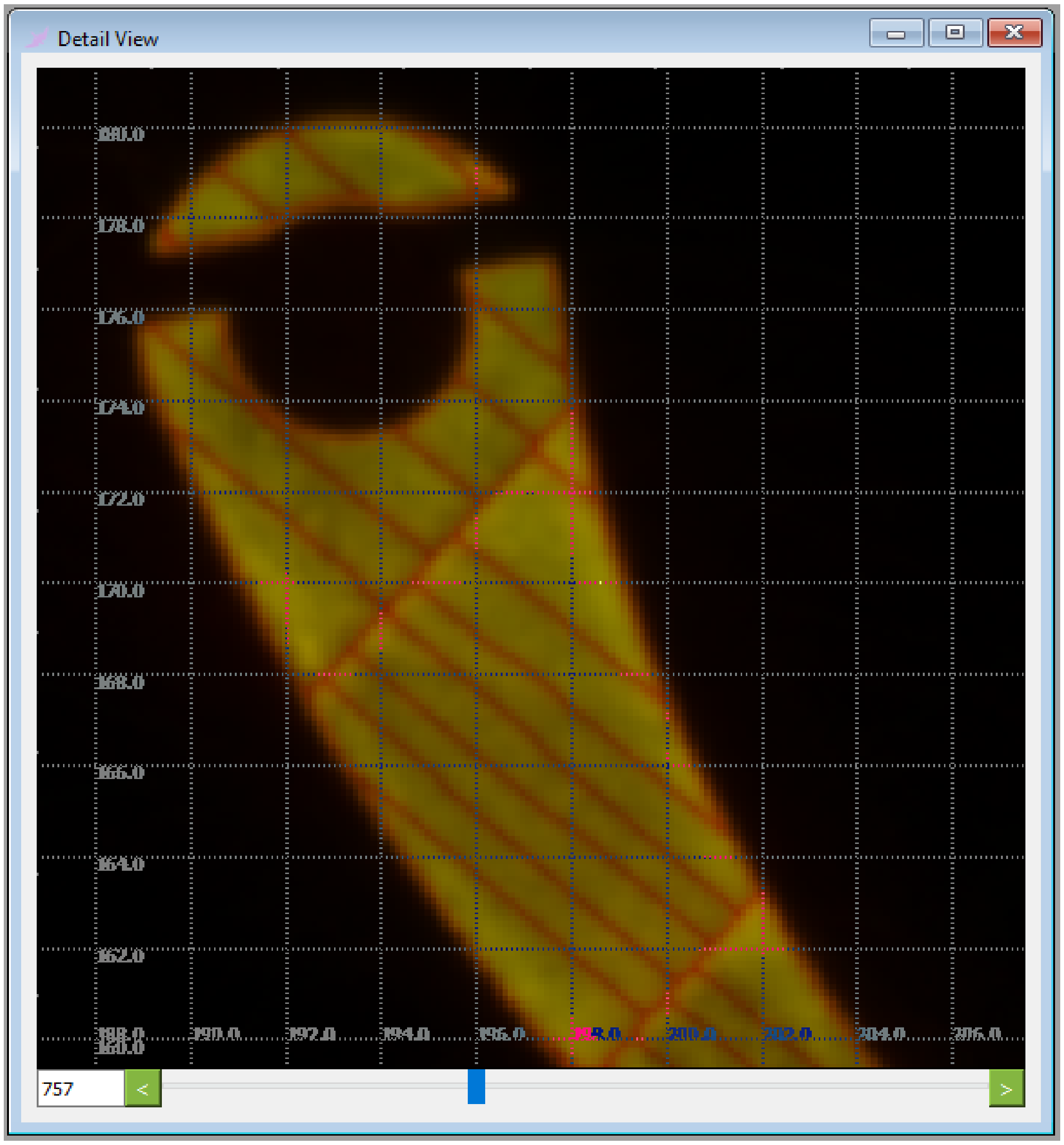


Figure 9. Details view.

This view provides the following features:

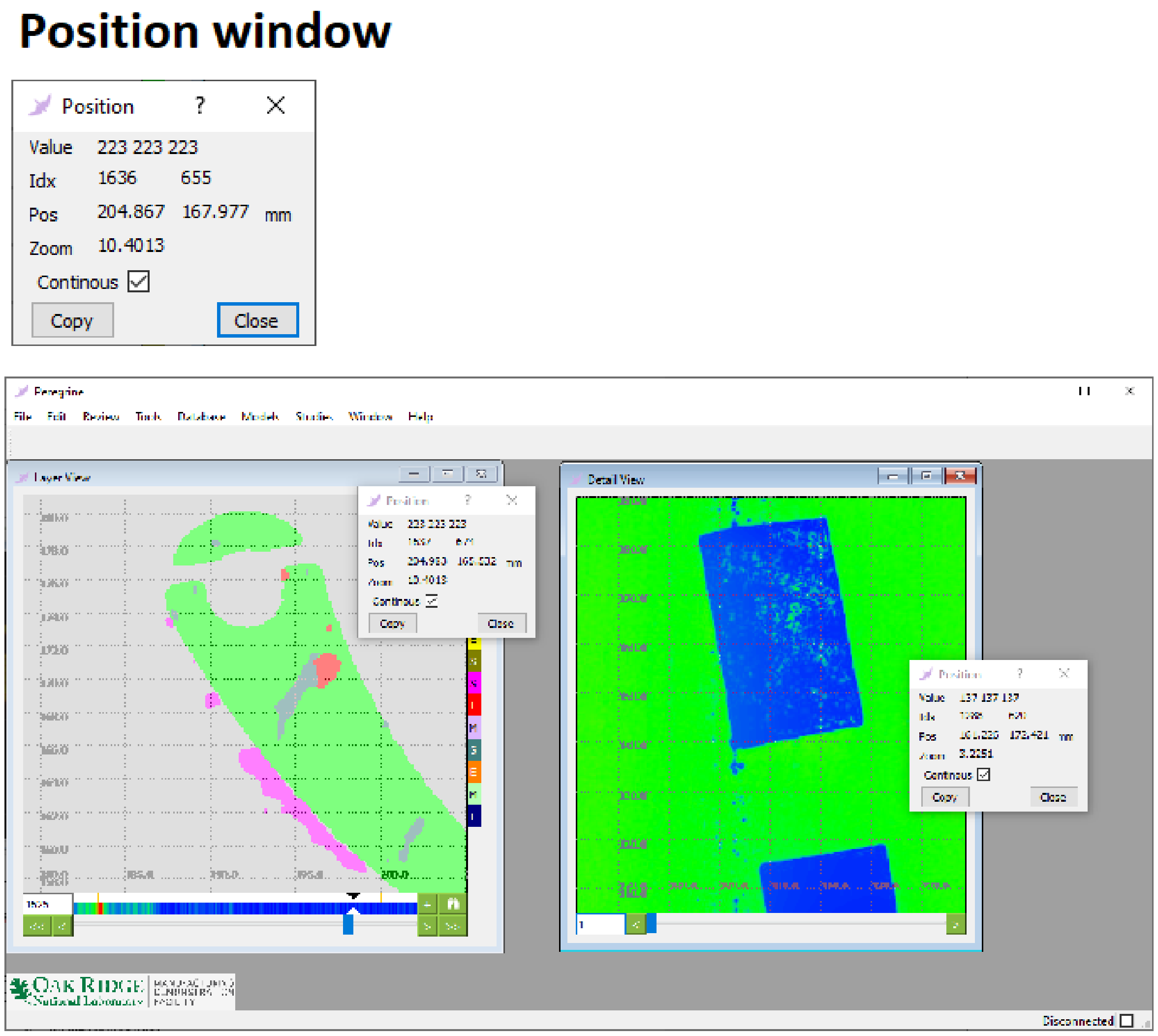
- Main graph displays the actual data
- Layer Selector allows the user to select the layer

This view can also display two satellite windows: the “Position” window and the “Settings” window.

This view uses the multi-channels with grid version of the main graph widget.

2.6.5 Position Window

The Position window (Figure 10) is a satellite window to both the Anomalies view and the Details view. It provides position data on the location of the cursor. This window can be updated continuously as the mouse moves or discontinuously, updating only when the user clicks the mouse. The position data can be exported to the system clipboard by pressing the copy button.



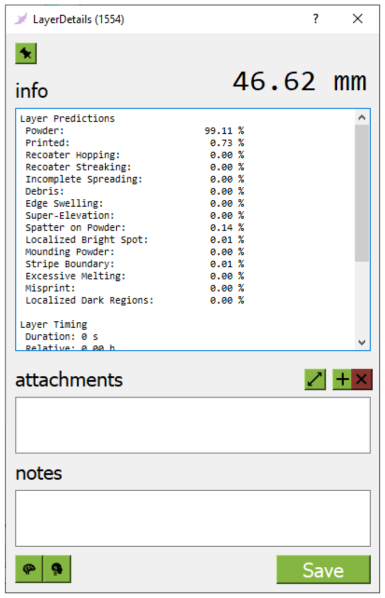
Value 223 223 223
Idx 1636 655
Pos 204.867 167.977 mm
Zoom 10.4013
Continous ☑

Figure 10. Position window.

2.6.6 Layer Details Window

The Layer Details window (Figure 11) is a satellite window for the Anomalies view. It provides information about a specific layer. If the window is not pinned to a layer, it updates continuously as the current layer changes. If it is pinned, the window displays information for the pinned layer.

Layer Details window



```
Layer Predictions
 Powder:                        99.11 %
 Printed:                        0.73 %
 Recoater Hopping:               0.00 %
 Recoater Streaking:             0.00 %
 Incomplete Spreading:           0.00 %
 Debris:                         0.00 %
 Edge Swelling:                  0.00 %
 Super-Elevation:                0.00 %
 Spatter on Powder:              0.14 %
 Localized Bright Spot:          0.01 %
 Mounding Powder:                0.00 %
 Stripe Boundary:                0.01 %
 Excessive Melting:              0.00 %
 Misprint:                       0.00 %
 Localized Dark Regions:         0.00 %

Layer Timing
 Duration: 0 s
 Relative: 0.00 h
```

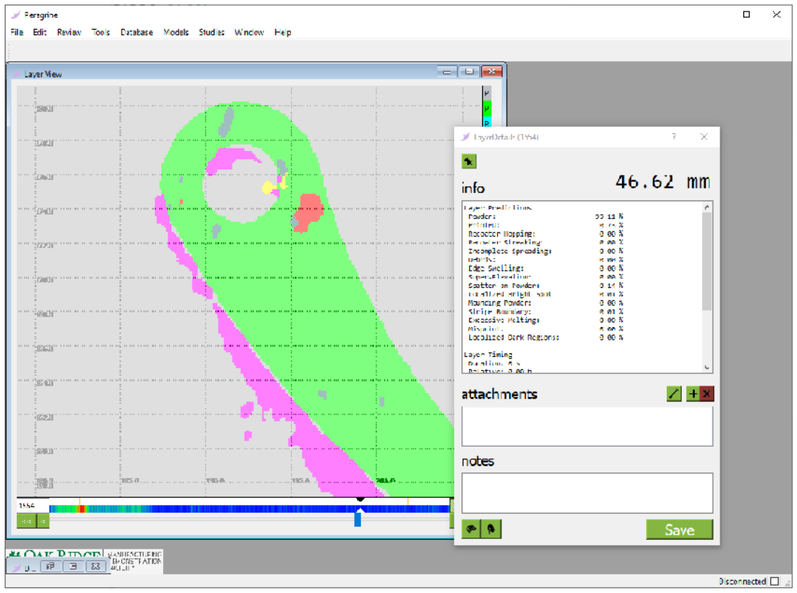


Figure 11. Layer Details window.

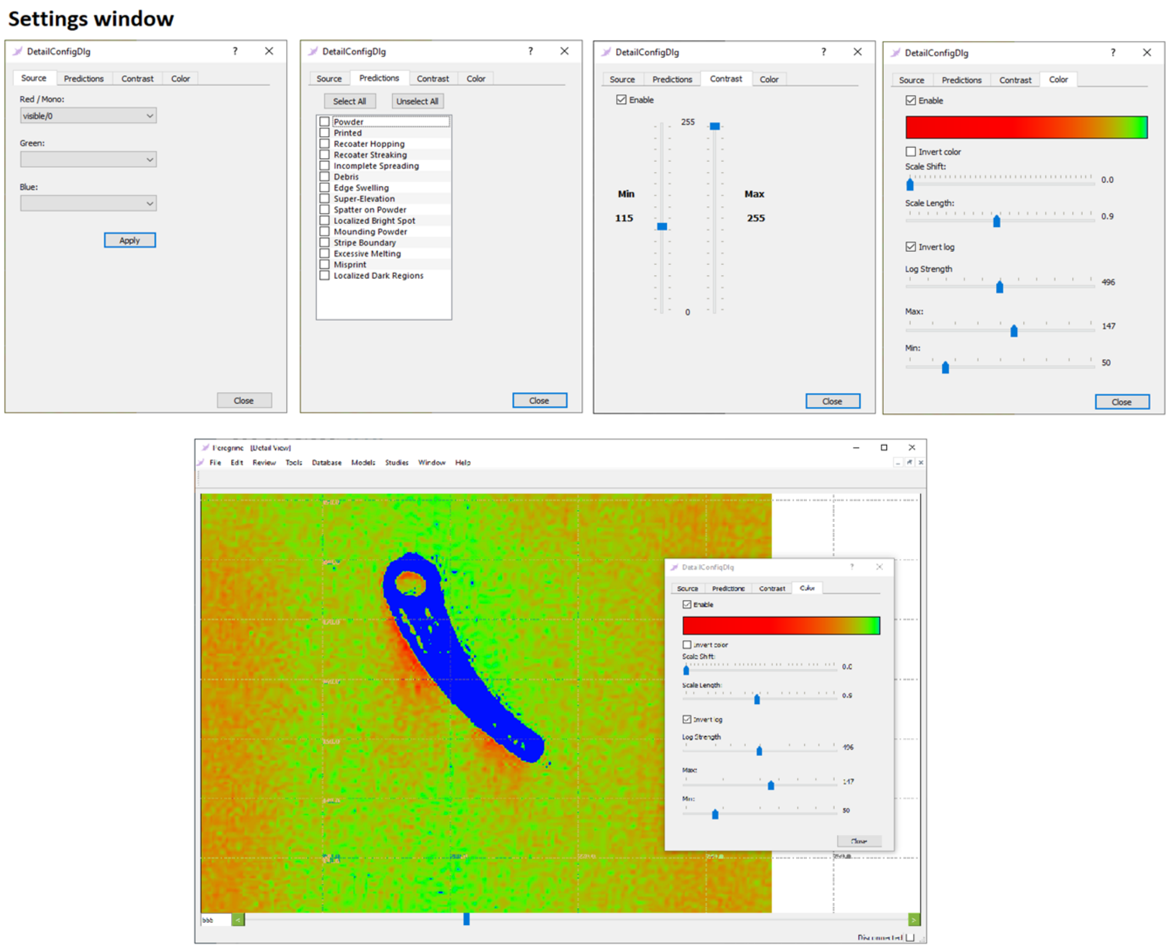
2.6.7 Settings Window

The Settings window () is a satellite window to the Details view. It provides controls over various features of a specific Details view. Changes are applied simultaneously as the user makes changes.

The Settings window controls features related to the following:

- Channels selection
- Anomalies overlay selection
- One channel dynamic range
- False color scale

Again, changes are applied instantly, giving the user instant feedback on the result of the changes



DetailConfigDlg — Source | Predictions | Contrast | Color

Red / Mono: visible/0

Green:

Blue:

Apply

Close

DetailConfigDlg — Predictions

Select All | Unselect All

- [ ] Powder
- [ ] Printed
- [ ] Recoater Hopping
- [ ] Recoater Streaking
- [ ] Incomplete Spreading
- [ ] Debris
- [ ] Edge Swelling
- [ ] Super-Elevation
- [ ] Spatter on Powder
- [ ] Localized Bright Spot
- [ ] Mounding Powder
- [ ] Stripe Boundary
- [ ] Excessive Melting
- [ ] Misprint
- [ ] Localized Dark Regions

Close

DetailConfigDlg — Contrast

[✓] Enable

255

Min 115

Max 255

0

Close

DetailConfigDlg — Color

[✓] Enable

[ ] Invert color

Scale Shift: 0.0

Scale Length: 0.9

[✓] Invert log

Log Strength 496

Max: 147

Min: 50

Close

Figure 12. Settings window.

2.7 SOURCE CODE REPOSITORY

The Git repository is located at:

https://code.ornl.gov/MDF_DataAnalytics/MDF-Peregrine-CPP

branch:

migration/phase1

The source code is located in the ORNL-managed GitLab repository. At this stage, access will be granted only to Advanced Materials and Manufacturing Technologies program members.

The source code for the version described in this report is available on this repository. Because this project is currently focused more on setting up the framework and adding features than on producing easily readable and maintainable code, the repository does not adhere to best programming practices.

At some point, the following steps will need to be taken:

- Remove dead code left over from the initial implementation
- Properly comment the code
- Lint the project

The Visual Studio solution compiles, runs, and is stable. However, no attempts have been made to generate a qmake or CMake solution from the Visual Studio project, nor has this project compiled or run the code on any platform other than Windows.

The code has been tested exclusively on Windows.

CONCLUSION

The current project provides a solid foundation for future expansion, demonstrating the feasibility of a highly flexible and responsive application for handling 3D printing data.

Compared to Python, C++ inherently offers faster execution speed, which is further enhanced by using multithreading whenever possible. Additionally, by carefully selecting efficient algorithms with favorable size and time complexity behaviors [e.g., $O(1) < O(\log(n)) < O(n) < O(n \log(n)) < O(n^2)$], this project ensured that the application remains responsive even when handling larger datasets.

Employing an MDI architecture (Figure 13) eliminates constraints on the location, size, and number of windows, allowing users to open as many windows as needed to effectively visualize the data.

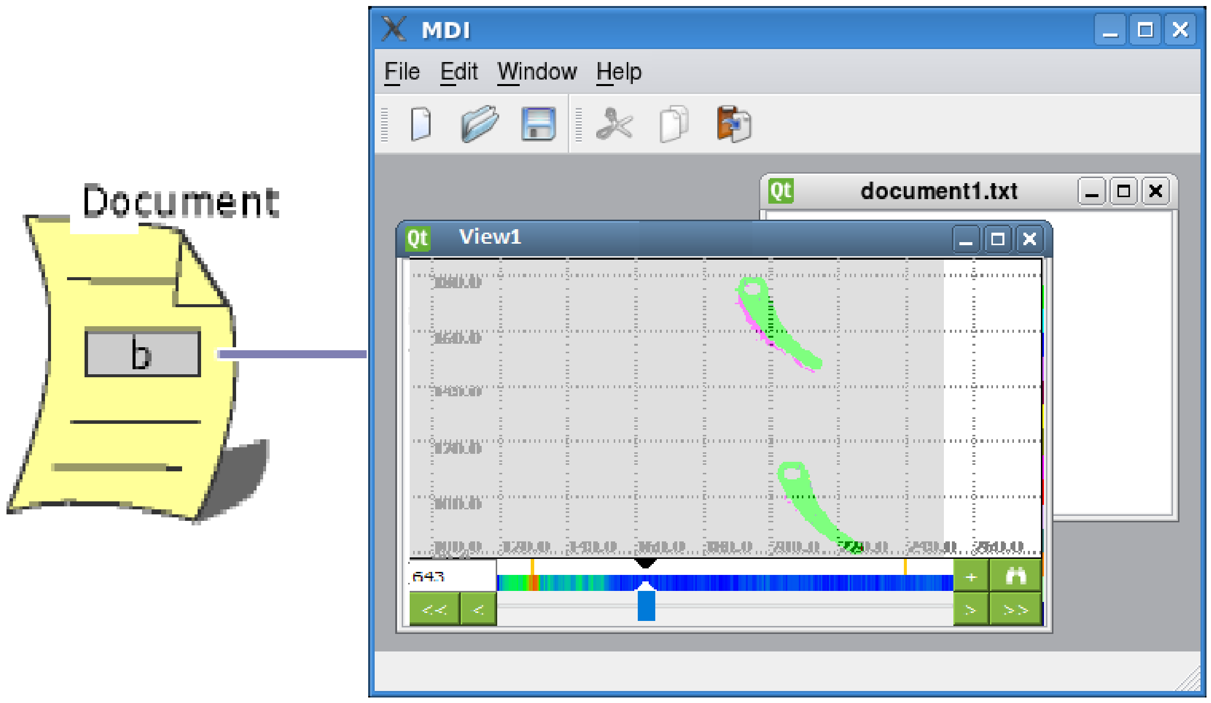


Figure 13. MDI architecture.

APPENDIX A. SYSTEM CONFIGURATION

The Peregrine C++ project is being developed on the following:

- CPU: I-7
- Solid-state drive (SSD): 3 TB SSD
- RAM: 80 Gb RAM
- GPU: Nvidia T1000
- Monitor: 4K at full resolution (3,840 × 2,160)

The recommended system is the following:

- CPU: I-7 or better
- SSD: 3 TB SSD or more
- RAM: 64 Gb RAM or more
- GPU: Any Nvidia GPU may help
- Monitor: 4K at full resolution (3,840 × 2,160)

The application should work on a lower resolution than 4K, but for the best results, a 4K monitor at full resolution is preferred.