HSItools documentation

Maurycy Żarczyński

Table of contents

Preface

This is a companion book to the HSItools R package, aiming at processing and visualizing hyperspectral scanning data.

Maurycy Żarczyński, David C. Edge, Nick P. McKay, and Paul D. Zander developed the package with the community's help.

The current requirements to run HSItools are as follows:

- R: >= 4.1.0 is necessary because we depend on the native R pipe and lambda functions introduced with R 4.1.0.
- Rtools (on Windows): >= 4.0 [link]
- QGIS (optionally but heavily encouraged): [link]

These downloads and installation take some time, depending on the Internet speed and your Operating System.

Installation

{HSItools} is currently in development and available only from GitHub.

> Package installation
>
> We are using {pak} for installation from remote sources.

```r
# Install pak if necessary
install.packages("pak")

# Install HSItools
pak::pak("mzarowka/HSItools")
```

> ! Terra
>
> Because the latest version of {terra}, which does most of the raster heavy-lifting, has a bug resulting in flipped images, we must install older version.

```r
# Install older version from CRAN
pak::pak("terra@1.7-78")
```

QGIS

In our workflow we delegate some of the work to GIS environment, especially marking the raster files, selecting regions and so on. We strongly suggest an Open Source tool like QGIS, however any GIS software, such as ESRI ArcGIS Pro should suffice.

You can find the latest version of QGIS here: https://www.qgis.org

1 Initial state

HSItools offers an easy way to preprocess Specim data. However, if the data follows the same rules, it can be generalized to the broader workflow.

1.1 Data structure

Data should be structured as follows:

– NAME <directory>

—- capture <directory>

—— DARKREF_NAME.hdr

—— DARKREF_NAME.log

—— DARKREF_NAME.raw

—— NAME.hdr

—— NAME.log

—— NAME.raw

—— WHITEREF_NAME.hdr

—— WHITEREF_NAME.log

—— WHITEREF_NAME.raw

However, if necessary, you can select appropriate files on your own. Files with the extension .raw are data files, while files with the .hdr extension are header files that contain essential information, such as the number of pixels, facilitating data reading.

1.2 Startup

Start by loading all necessary packages{HSItools} will load essential functions from the namespace, but to be sure, call all packages.

```
# Load packages
library(HSItools)
library(terra)
library(tidyterra)
library(dplyr)
library(tidyr)
library(ggplot2)
library(signal)
library(sf)
```

Hyperspectral data gets quite large. It is a good practice to process data stored on an SSD drive separate from your Operating System (OS). While data is stored on a separate drive, it is beneficial to instruct {terra} to store temporary data on a separate drive, too. Adjust this according to your OS. Here, we are using a non-system drive on Windows 11. We set the maximum allowed RAM to a high value of 0.9. It would be best to decide according to the available memory and OS requirements.

```
# Set tempdir
terra::terraOptions(tempdir = "D:/", memmax = 0.9)
```

First, this is specific only for workflow, where you'd like to calculate reflectance from the Shiny output. We must set the working directory (otherwise, we discourage this approach in R).

```
# Set working direcotory
setwd("C:/GitHub/data/cake/")
```

2 Shiny app

Our shiny app allows quick choice of data, settings, regions of interest (ROI), and depth calibration. Here, we walk through the entire app, screen by screen.

2.1 Screen 1: Initial settings

On this screen, you have to make an initial choice. First, you need to decide whether to normalize data. If you have a reflectance file from other software, like Lumo®, you probably do not need to normalize the file from the beginning—this is one of the most time-consuming processes.

Suppose you decide that you need to normalize your data. In that case, you can select other integration times for your white and dark references if you scanned your target with different settings for target and references. This can happen if you were worried about white reference overexposure, whereas your target was very dark. Finally, you can select some proposed HSI indices from the defaults.

> **Indices**
>
> At the moment, selecting indices will not do anything in particular.

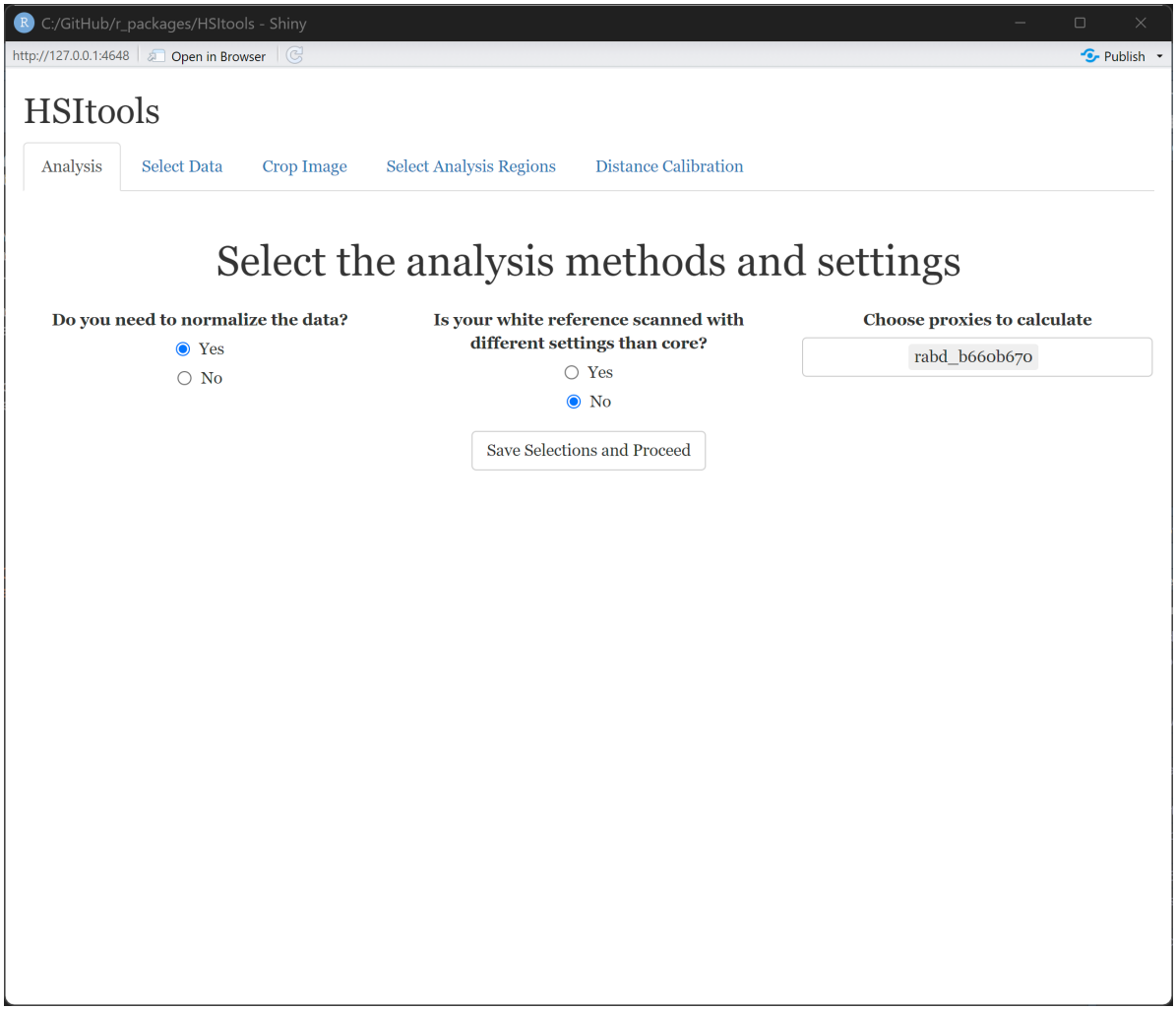


Figure 2.1: Shiny – first screen, basic options.

2.2 Screen 2: Data choice

This screen lets you decide whether to use your data or the provided example.

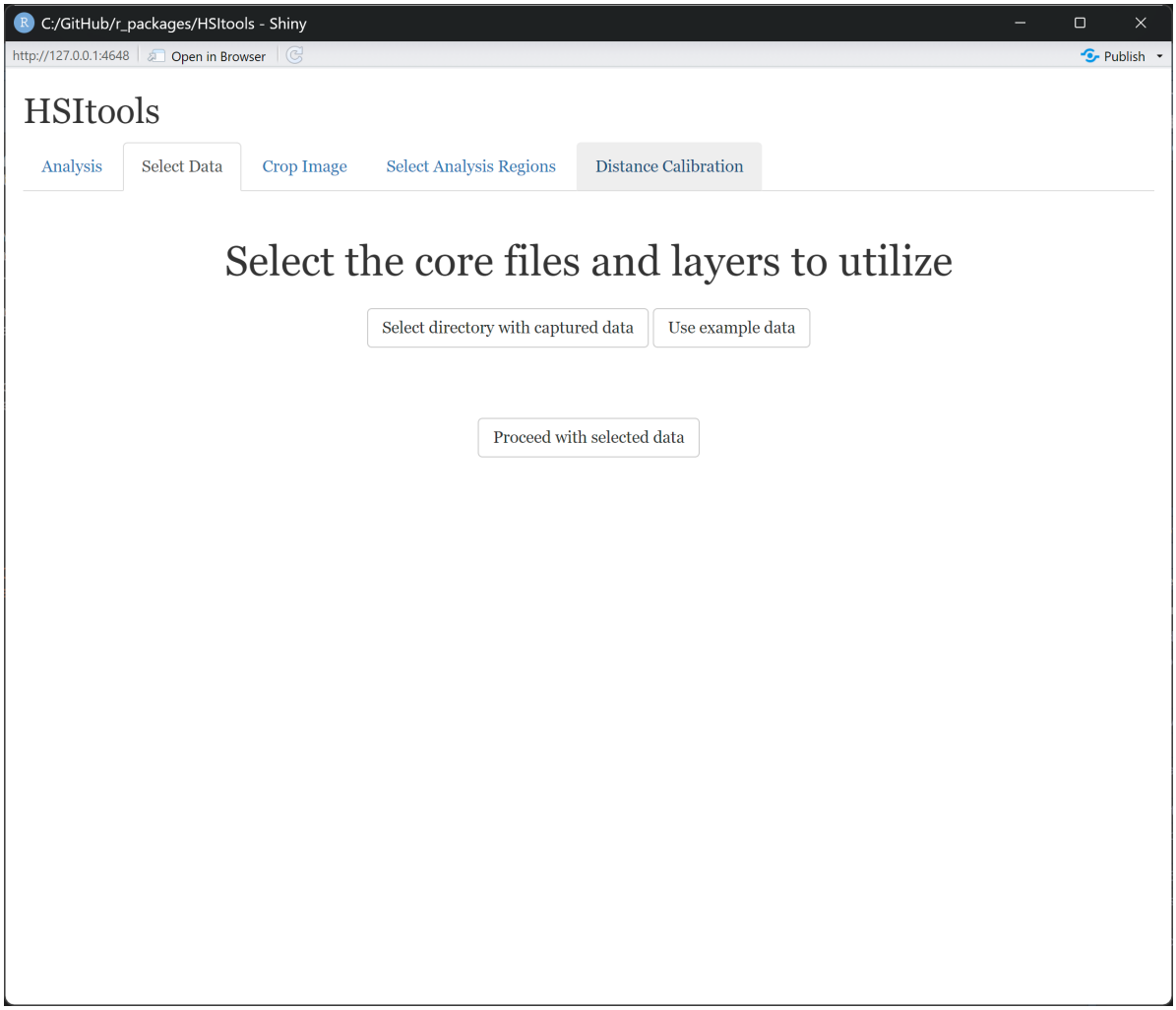


Figure 2.2: Shiny – data choice.

You can select your data from your drive once you use it. Be sure that it is stored on the SSD drive. Here, it is essential that you do not go all the way to the capture folder but rather point to the root directory of your selected scan.

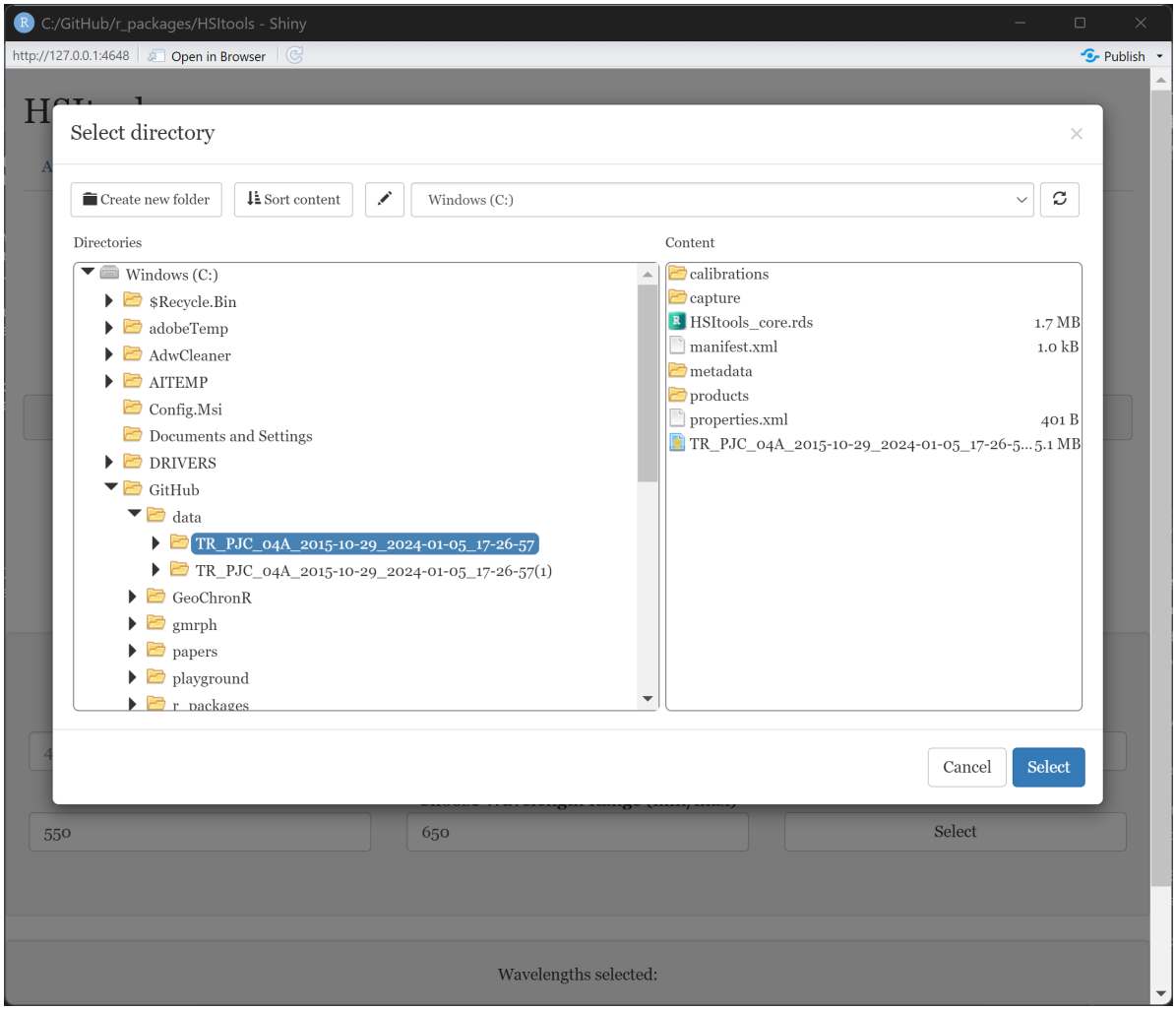


Figure 2.3: Shiny – data selection.

Once you've selected data, you can choose the raster files, bypassing our heuristics and default search patterns in the "capture" directory. At this point, you'll also get some basic information on your raster dimensions.

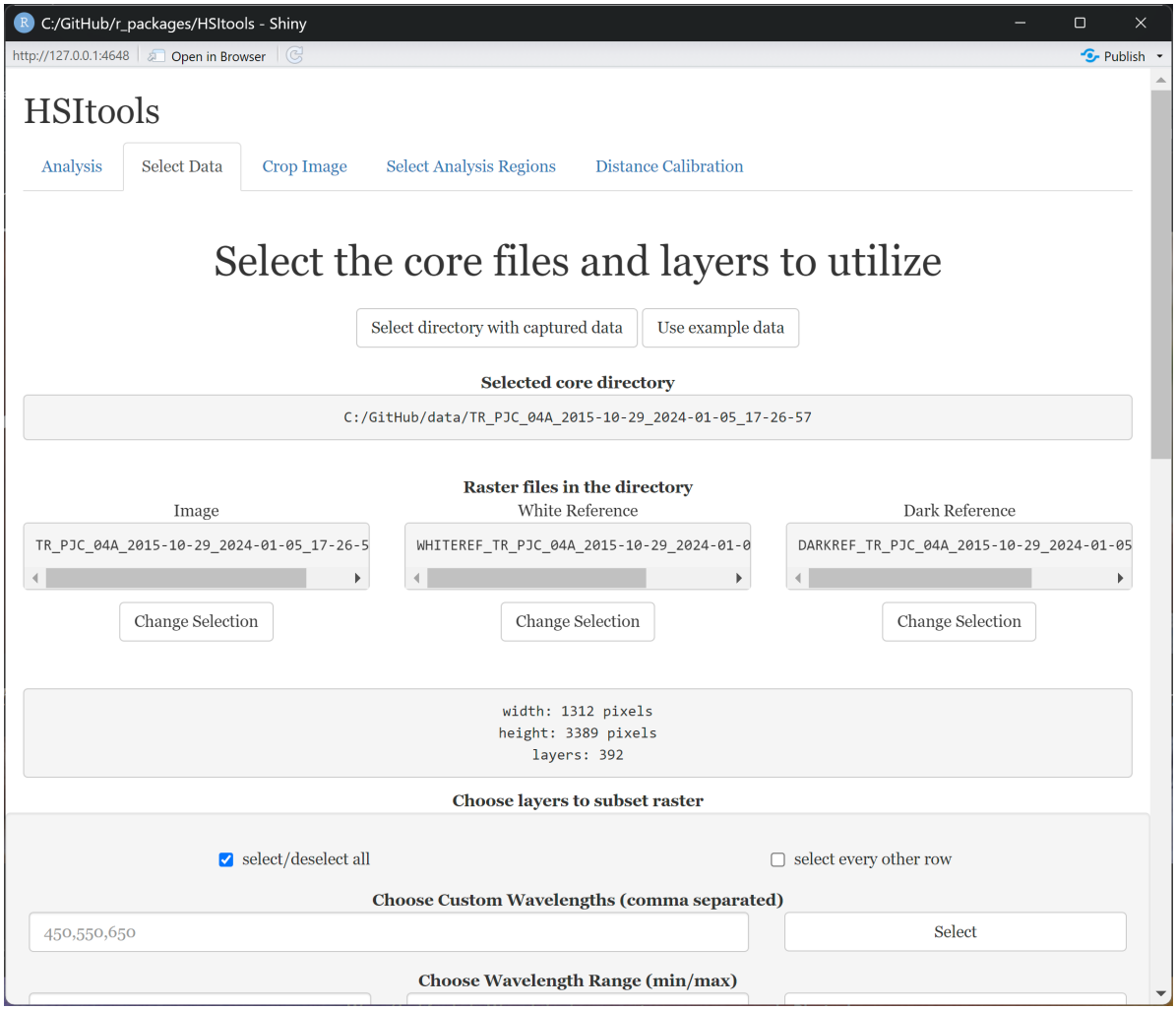


Figure 2.4: Shiny – data selection continued.

Once you're happy with your data choice, select layers (bands) to work with.

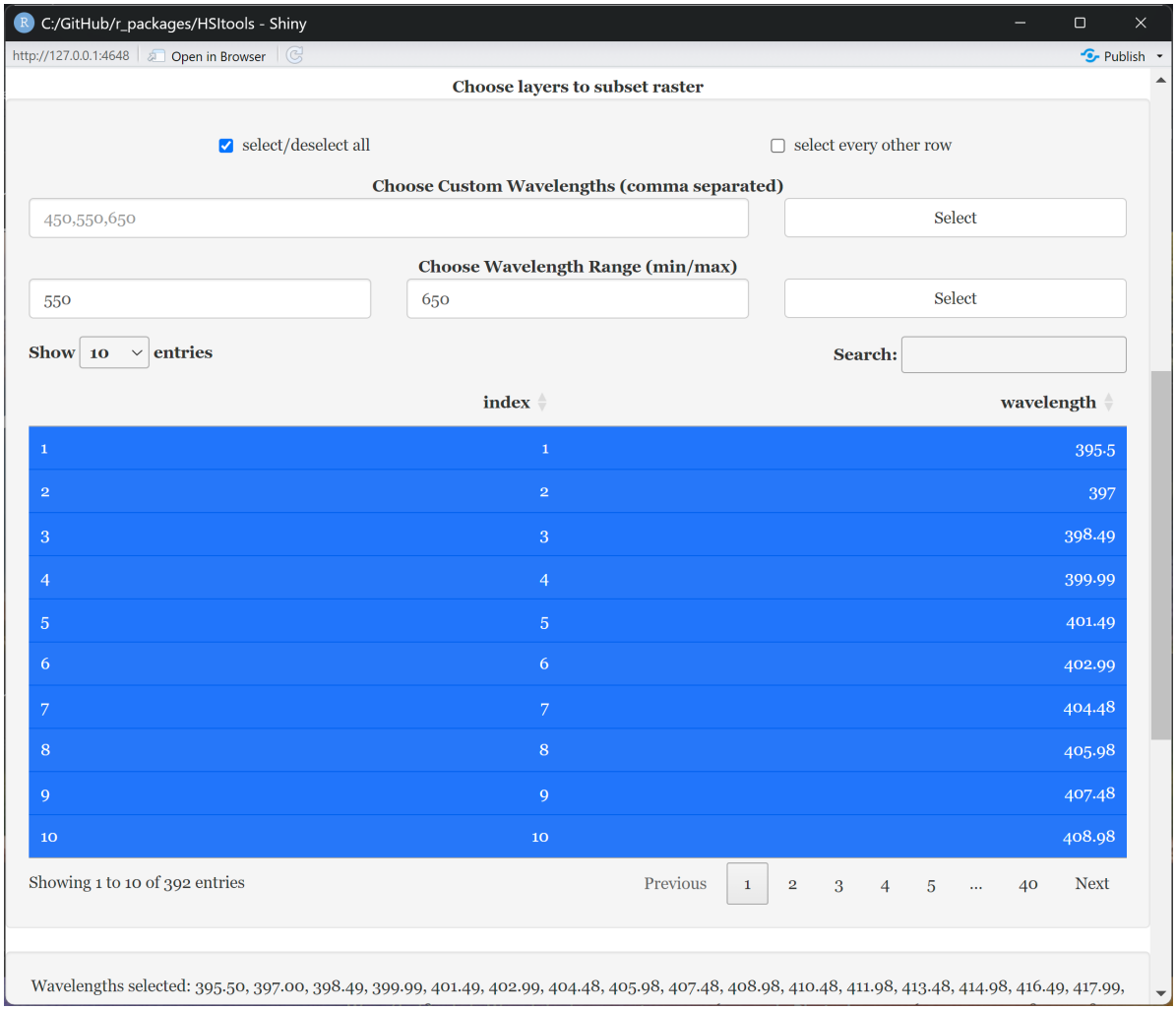


Figure 2.5: Shiny – data selection, layers.

If you're happy with your choices, scroll down and proceed.

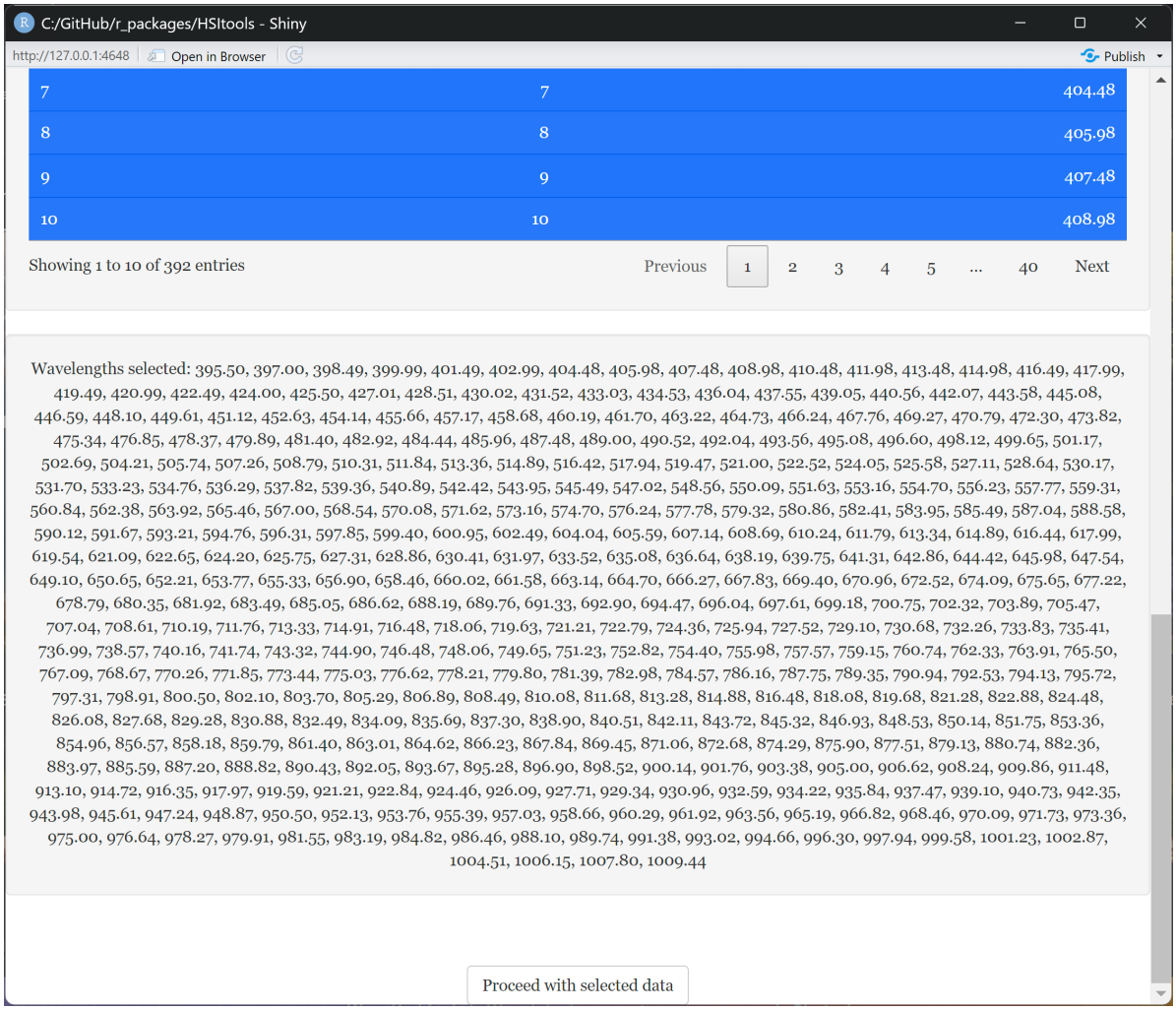


Figure 2.6: Shiny – data selection, finish.

2.3 Screen 3: Cropping

> **!** Raster preview
>
> Some scans will look "bad" on this and the next screen. It is still your data. We're working on it! :)

Here, you can crop your data (raster). You can see the bounding box of the cropped data. If it helps, you can change the size of the image. Leave a pixel or two on the edges; if pushed to the extremes, it can mess up the cropping. Once you're happy, accept the crop and proceed.

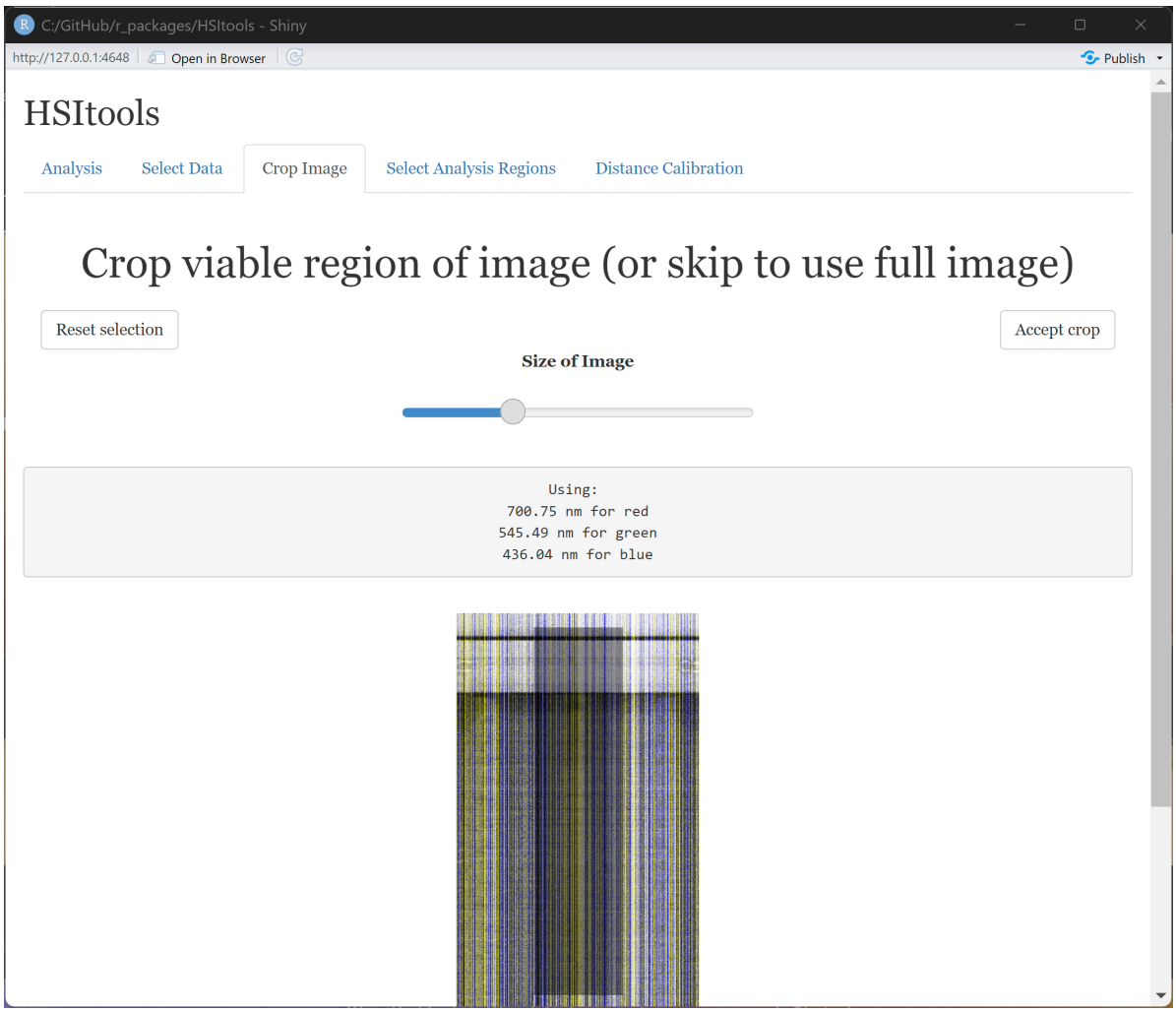


Figure 2.7: Shiny – cropping.

2.4 Screen 4: ROI selection

Here, you can select the regions of interest (ROI) you want to work over rather than calculating the entire data. These ROIs are strictly rectangular, so we prefer the GIS approach with masking. Please select the region, then add it using the button. Repeat for as many regions as you need. On the right, you have a zoom of your selection. Once you're happy with your ROIs, accept and proceed.

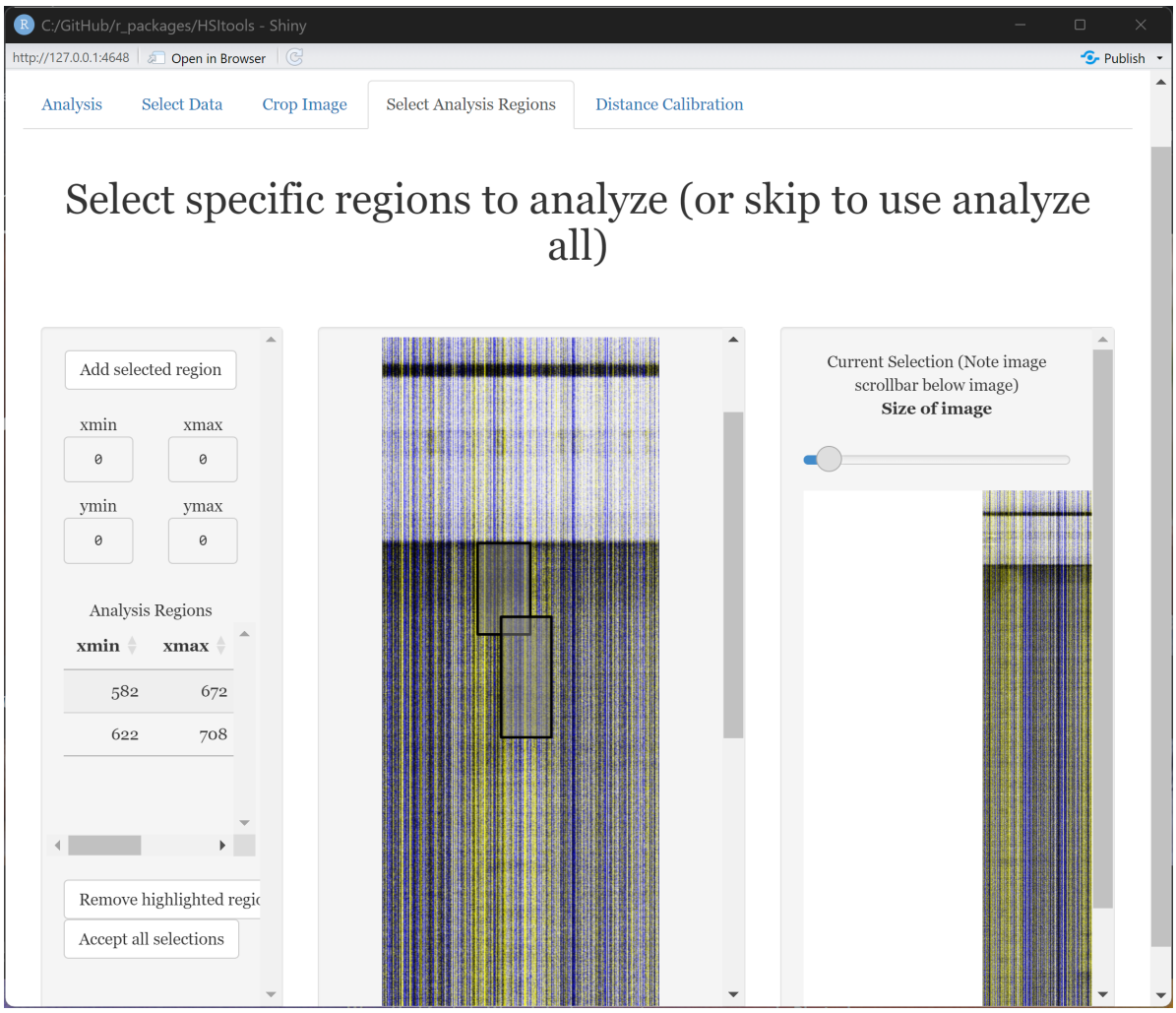


Figure 2.8: Shiny - ROIs selection.

2.5 Screen 5: Calibration

> **!** RGB preview
>
> Here, the captured data looks as it should!

Finally, it is time to calibrate your data.

First, we calibrate the distance. Simply put, we want to know the actual distance in one pixel. For this, we use the tape measure and put the two markers. Then, we read the depth from

the tape—the further the distance between the points, the better. We use only 20 mm, so it fits on the screen. Our equation takes into account both the distance and angle.

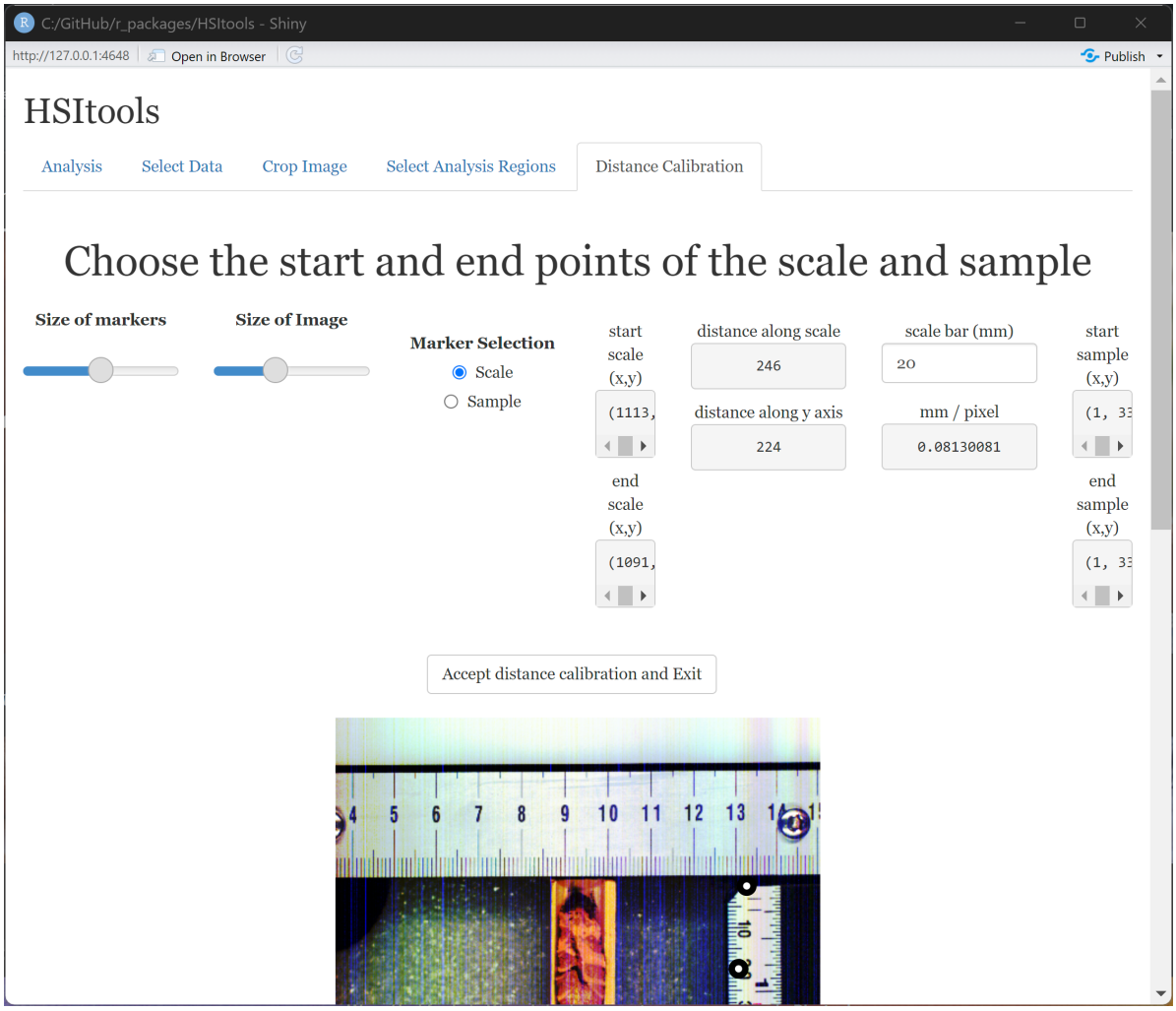


Figure 2.9: Shiny – depth calibration.

Finally, we put in the second set of markers, which tracks the actual beginning and end of the scanned sample—the true-zero is essential here.

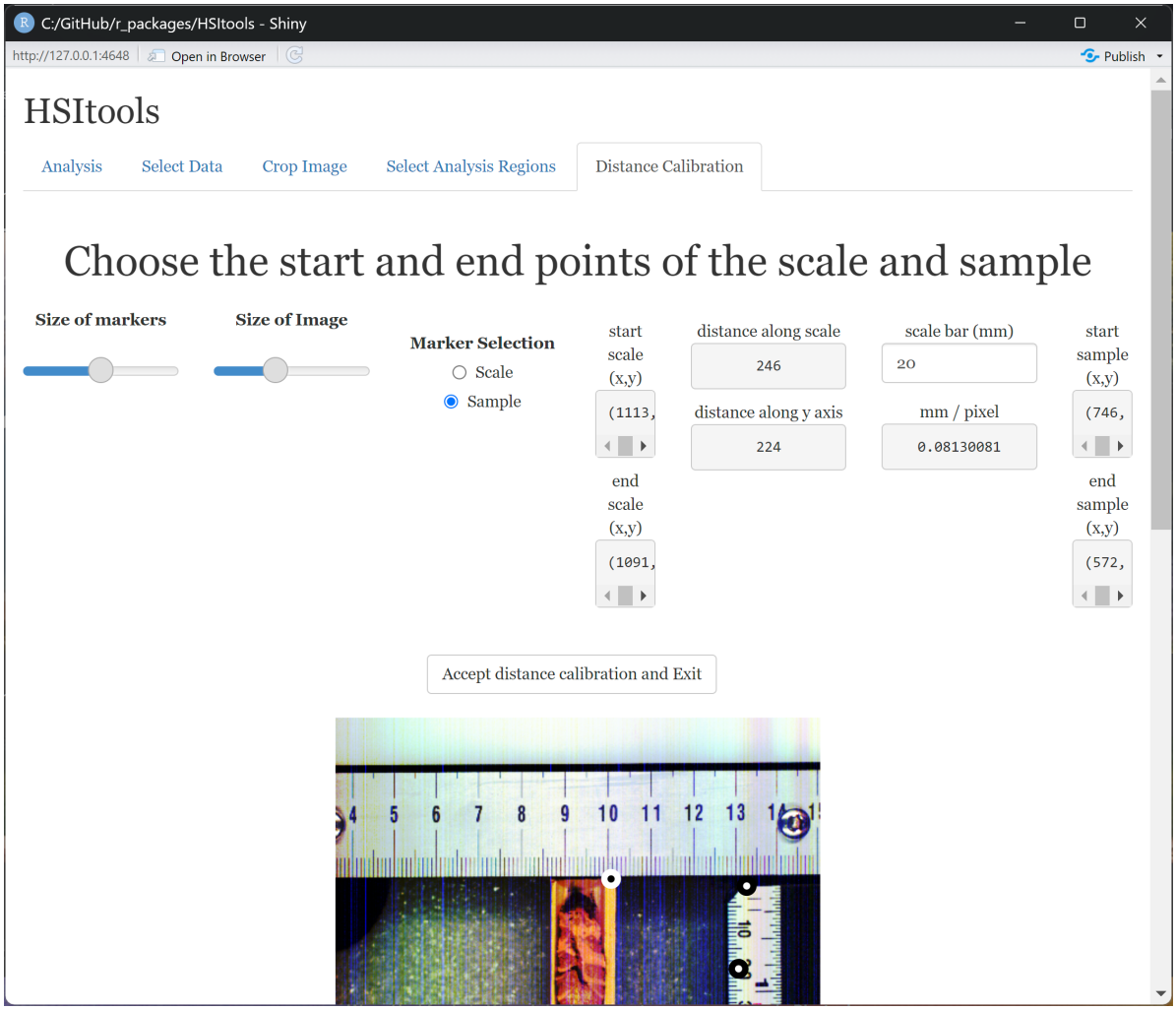


Figure 2.10: Shiny – true-zero calibration.

Once you select both sets of markers, accept and exit the app.

3 Preprocessing

3.1 Normalization

Before any spectral indices and properties are calculated, normalizing the data and expressing it as a reflectance is necessary. Here, reflectance is a fraction of the signal between the dark and white references acquired during or before the scan.

Normalization is achieved by following the equation:

$$reflectance = (d_{raw} - dark_{ref})/(white_{ref} - dark_{ref})$$

where:

reflectance: normalized data cube;

d_{raw}: raw captured data;

$dark_{ref}$: dark reference data;

$white_{ref}$: white reference data.

Which can be modified for different acquisition times for white reference:

$$reflectance = (d_{raw} - dark_{ref})/(white_{ref} - dark_{ref}) * t_{int}white/t_{int}sample$$

where:

t_{int}white: integration time of white reference;

t_{int}sample: integration time of captured data (sample).

3.1.1 Normalization with Shiny output

If Shiny GUI was used for data selection, cropping, and calibration, then it would be easy to pass Shiny's output to the normalization routine. The normalized files will be written into your data's products directory.

First, this is specific only for workflow, where you'd like to calculate reflectance from the Shiny output. We must set the working directory (otherwise, we discourage this approach in R).

```
# Set working direcotory
setwd("C:/GitHub/data/cake/")
```

You either have the `HSItools::run_core()` output in memory or you need to read it from disk.

For some reason, once read out from the disk, the .rds file drops the directory. We need to fix it with code.

```
# Read from disk
hsi_tools_core <- readRDS("Achive_half_WR_240212-111356/HSItools_core.rds")

# Add the directory back
hsi_tools_core$directory <- "Achive_half_WR_240212-111356"
```

Then, use the standard function to calculate the reflectance.

```
# Create normalized reflectance file
reflectance <- hsi_tools_core |>                                    ①
  HSItools::prepare_core()                                          ②
```

① The Shiny GUI output.
② Normalization function.

If you used different integration times for your sample and white references, provide this information in the function.

```
# Create normalized reflectance file
reflectance <- hsi_tools_core |>                                    ①
  HSItools::prepare_core(                                           ②
    tints = 90,                                                     ③
    tintw = 30)                                                     ④
```

① The Shiny GUI output.

② Normalization function.
③ Integration time of sample (captured data).
④ Integration time of white reference.

> **Multiple cores**
>
> It is planned to optionally iterate over multiple directories using, for example, the {purrr} package. This will need to get rid of setwd() call.

3.1.2 Normalization of the directory (no Shiny output)

If no Shiny output is available and input is not produced by hand, the normalization routine can be run without it. In such a case, the entire capture data will be normalized. However, without Shiny output, it is harder to calibrate distances properly.

Here, we normalize all the captured data with all the available layers between 400 and 1000 nm.

```
# Store the path
path <- "data/my_core"

# Create normalized reflectance file
reflectance <- path |>                                                    ①
  HSItools::prepare_core(path = path, layers = c(400:1000), extent = "capture") ②
```

① Path to your core of choice, root directory. Don't go to the "capture" directory.
② Normalization function. The path is your core path, then the selection of layers (bands) and an explicit call informing that the entire capture extent should be normalized.

It is possible to iterate over multiple directories at once using the `{purrr}` package.

```
# Store the paths
paths <- list(                                                            ①
  core_1 = "data/my_core_1",
  core_2 = "data/my_core_2")

# Or easier, with listing
paths <- fs::dir_ls("data")                                               ②

# Create normalized reflectance files
reflectance <- paths |>                                                   ③
  purrr::map(\(i) HSItools::prepare_core(
```

```
    path = i,
    layers = c(400:1000),
    extent = "capture")) ④
```

① Create a list of directories by hand.
② Or list root directories in your higher-level directory.
③ Your paths.
④ Normalization function iterated over multiple directories.

3.2 Savitzky-Golay smoothing

Applying spectral smoothing, such as the Savitzky-Golay spectral filter (Savitzky and Golay 1964), prevents spurious, random peaks and troughs from significantly influencing the calculation results.

```
reflectance <- reflectance |> ①
  # Specify the file extension
  HSItools::filter_savgol() ②
```

① Reflectance data is either in memory or from a disk.
② Calculation function.

> **Speed**
>
> This function applies a smooth filter to every pixel. Depending on the size of your data, it will take considerable time to run.

3.3 Median filter

You might be interested in smoothing your data spatially to remove some of the noise in the XY domain.

```
reflectance <- reflectance |> ①
  # Specify the file extension
  HSItools::filter_median() ②
```

① Reflectance data is either in memory or from a disk.
② Calculation function. You can set the window (defaults to 3 by 3).

> **Speed**
>
> This function applies a focal median filter to every pixel. Depending on the size of your data, it will take considerable time to run. More than the Savitzky-Golay. Consider running it on the calculated index rasters and ROIs.

3.4 Continuum removal

We can process our data further by removing the continuum, which follows the rule of dividing the spectrum by its bounding box. This way, the spectrum becomes flatter.

```
reflectance_cr <- reflectance |>                    ①
  HSItools::remove_continuum()                      ②
```

① Reflectance data is either in memory or from a disk.
② Calculation function.

3.5 Preview

For later use, generate a RGB (CIR, NIR) preview.

```
rgb <- reflectance |>                                                   ①
  HSItools::stretch_raster_full(type = "RGB", extension = "tif")        ②
```

① Reflectance data is either in memory or from a disk.
② Stretching function, with type set to RGB (also accepts CIR and NIR) and an extension, works with png, too.

4 Regions of interest

Selecting regions of interest (ROI) is a crucial task that requires deciding which parts of the spatial dataset will be analyzed. With HSItools, we follow two paths to select the data. The short one is based solely on the Shiny output and is limited, and the longer one is based on the Geographic Information System (GIS Software). In the spirit of the free and open source, we suggest QGIS, but other apps, such as ArcGIS Pro, allow the same functionality. What's important is that using GIS will enable you to freely draw shapes for data masking, a technique that helps you to focus on specific areas of the dataset while excluding others – more on this later.

4.1 Short path – The Shiny Path

If you opt for the Shiny path, you will work with ROIs selected in the Shiny app. These ROIs are strictly rectangular, and you don't have an immediate option to mask the data. However, you can still do it later, as you can mask any raster data. The selected regions are available within the HSItools_core.rds file or stored as an in-memory object.

```
# Get the ROIs
rois <- hsi_tools_core$analysisRegions
```

4.2 Long path – The GIS Path

Here, we assume we are at least somewhat familiar with the GIS environment.

> QGIS project
>
> It is a good idea to save the QGIS project file so you can keep the placement and customization of your layers. We suggest keeping the project name the same as your captured data.

4.2.1 QGIS

For speed, we load the full-resolution RGB preview generated previously with the `stretch_raster_full()` function, preferably a .tif file. Simply dragging and dropping into the empty window is enough. In the options of your new raster layer, you can increase your data's brightness or contrast, making drawing easier.

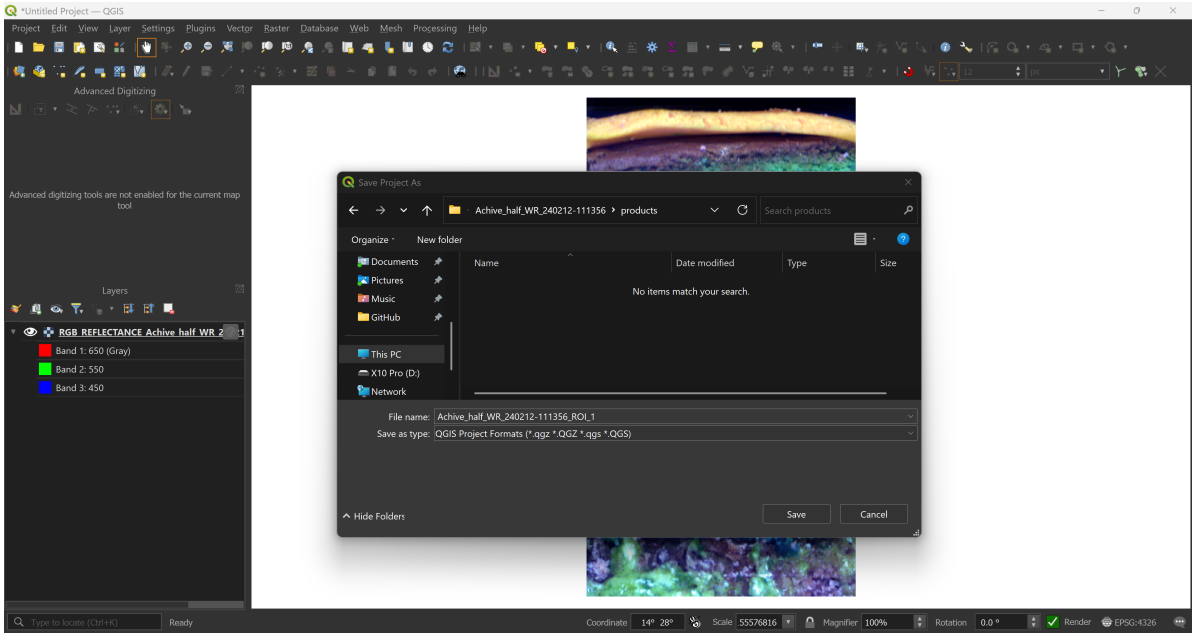


Figure 4.1: QGIS - saving the project.

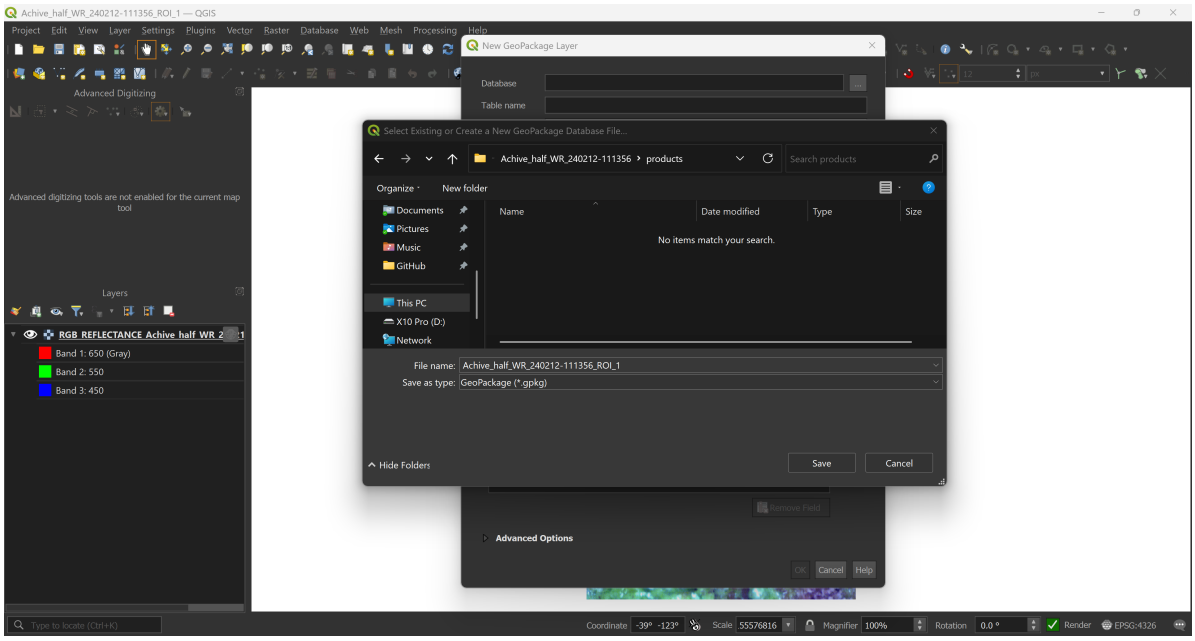


Figure 4.2: QGIS - creating a new GeoPackage.

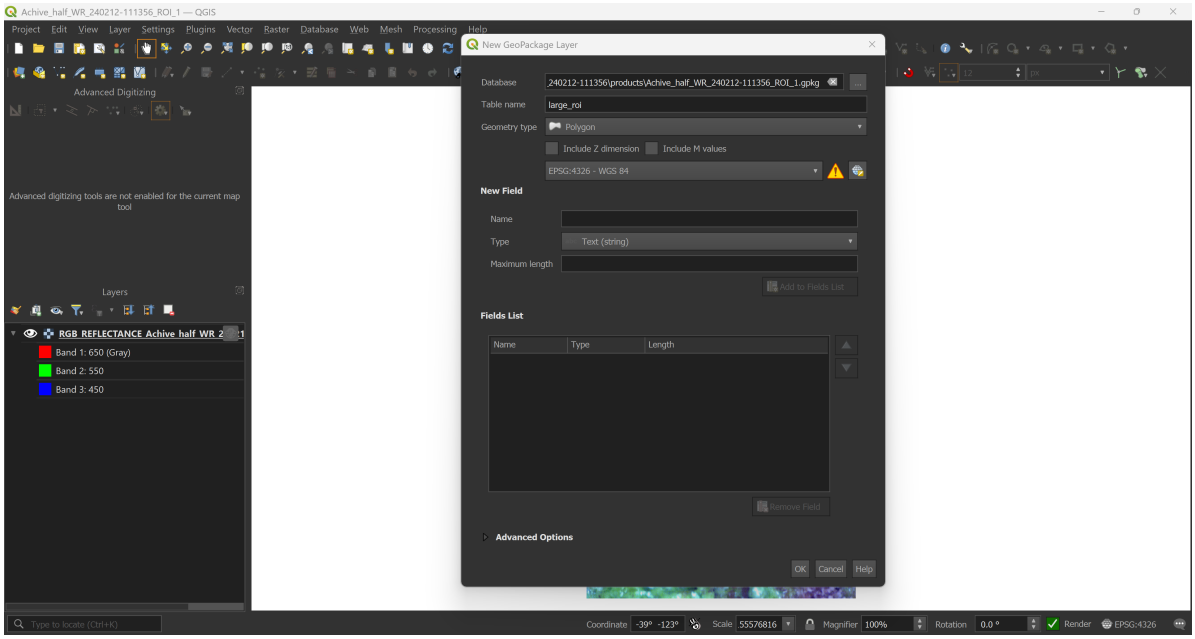


Figure 4.3: QGIS - creatin a new GeoPackage, continued. Adding a table for big ROI.

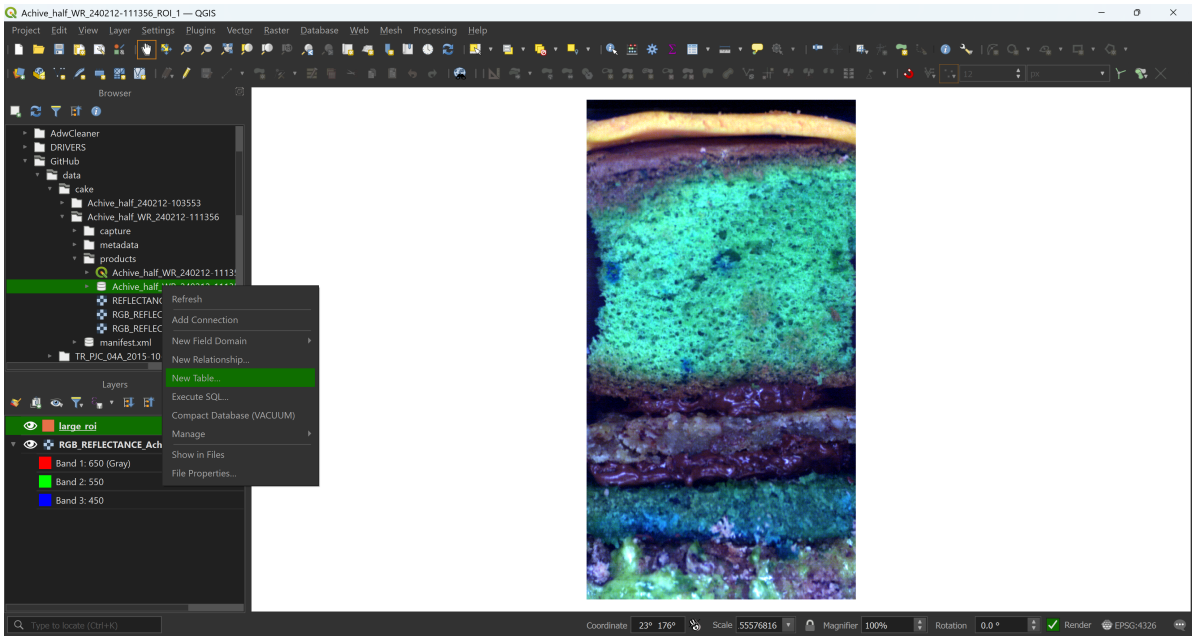


Figure 4.4: QGIS - adding new tables to a GeoPackage (small ROI, samples and masks).

Once you're happy with your project and data presentation, start drawing. Here we have a large ROI, covering most of the core, a smaller ROI that we will use to extract the data, even smaller ROIs for sample extraction and masks covering parts of the captured data that should be excluded from the analysis.

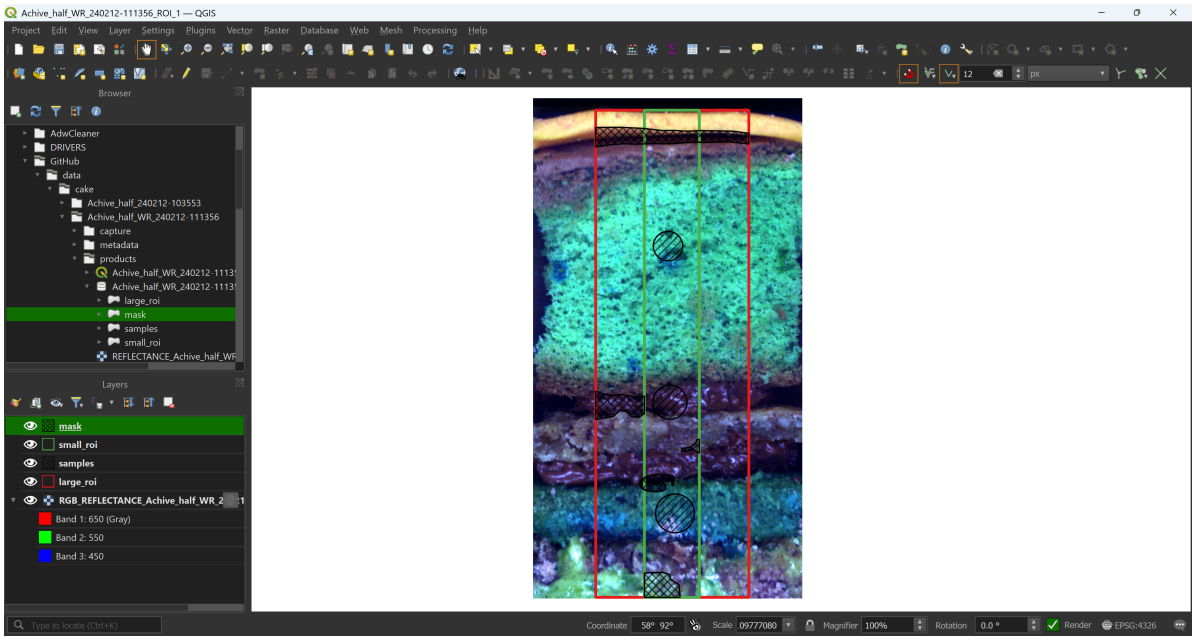


Figure 4.5: QGIS - an example of large and small ROIs, sample markers and masks.

4.2.2 Reading data in R

We now have all our ROIs. If you need to correct them, add or remove some polygons it is fine, as long as you do not open the GeoPackage file both in QGIS and R at the same time. We keep these ROIs for later.

```
# Read data
# Large ROI
rois_lrg <- sf::read_sf("Achive_half_WR_240212-111356/products/Achive_half_WR_240212-111356_I

# Small ROI
rois_sml <- sf::read_sf("Achive_half_WR_240212-111356/products/Achive_half_WR_240212-111356_I

# Samples ROIs
rois_smp <- sf::read_sf("Achive_half_WR_240212-111356/products/Achive_half_WR_240212-111356_I

# Mask ROIs
rois_msk <- sf::read_sf("Achive_half_WR_240212-111356/products/Achive_half_WR_240212-111356_I
```

5 Indices

HSItools offers a way to calculate a few of the most common indices classes used in paleoenvironmental investigations.

5.1 Masking

Before we calculate the indices we can mask the regions of the data that should not be used in the calculations. Here we assume, that we have the mask ROIs available in the environment.

```
# Mask the data
reflectance_msk <- reflectance |>                    ①
  # Use terra
  terra::mask(                                       ②
    roi_msk,
    inverse = TRUE)                                  ③
```

① Reflectance data is either in memory or from a disk.
② {terra} masking funciton.
③ Object with masks.

In every subsequent step it is your choice if you'd like to calculate indices on the full reflectance or the masked one.

5.2 Mean reflectance (Rmean)

The most straightforward index is the mean of the reflectance values of all selected wavelengths within a given pixel. It reflects overall changes in the darkness or brightness of the captured specimen. It can be used to normalize other indices or filter the data, for example by removing too dark pixels.

```
# Calculate rmean
rmean <- reflectance |>                              ①
  HSItools::calculate_rmean()                        ②
```

① Reflectance data is either in memory or from a disk.
② Calculation function.

You can also specify the extent of calculation.

```
# Calculate rmean
rmean <- reflectance |>                                    ①
  HSItools::calculate_rmean(                               ②
    extent = roi_sml[1, ]                                  ③
  )
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Definition of the extent.

You can use this approach within every subsequent function for calculation of an index.

5.3 Relative Absorption Band Depth (RABD)

A common index, where typically values starting at 1.0 indicate reflectance of a given substance.

The RABD calculation has variations, but the results are generally not drastically different. You can use predefined values or provide them manually.

The output's name informs you about the calculated proxy and additional modifications to the reflectance file; here, we calculated it with Savitzky-Golay smoothed reflectance. Let's calculate one of the most common indices to estimate the total chloropigments-*a*: $\text{RABD}_{660:670}$.

5.3.1 Variant 1 – "max"

In this variant, a minimum reflectance is found in the trough for each pixel and flexibly used for calculations.

```
# Calculate RABD
rabd_max <- reflectance |>                                 ①
  HSItools::calculate_rabd(                                ②
    edges = c(590, 730),                                   ③
    trough = c(660:670),                                   ④
    rabd_name = "rabd660670",                              ⑤
    rabd_type = "max")                                     ⑥
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Edges of the trough, the broader scope.
④ Trough of interest, a narrower scope to find the reflectance value.
⑤ Name of the index to be stored in the raster data.
⑥ Type of the RABD calculation.

5.3.2 Variant 2 – "strict"

This classic variant supplies a specific wavelength to calculate RABD for every pixel. Therefore, it is RABD_{665}.

```
# Calculate RABD
rabd_strict <- reflectance |>
  HSItools::calculate_rabd(
    edges = c(590, 730),
    trough = 665,
    rabd_name = "rabd665",
    rabd_type = "strict")
```

5.3.3 Variant 3 – "midpoint"

This is variant 2 (strict), with the added shortcut of always finding the middle point between the through edges—a convenience shortcut for some. Here, it equals RABD_{665}.

```
# Calculate RABD
rabd_midpoint <- reflectance |>                    ①
  HSItools::calculate_rabd(                        ②
    edges = c(590, 730),                           ③
    trough = c(660:670),                           ④
    rabd_name = "rabd660670",                      ⑤
    rabd_type = "mid")                             ⑥
```

5.4 Relative Absorption Band Area - ToDo

> **ToDo**
>
> This is a feature yet to be added.

```r
# Calculate RABA
raba <- reflectance |>                                          ①
  HSItools::calculate_raba(                                     ②
    edges = c(590, 730),                                        ③
    trough = c(660:670),                                        ④
    raba_name = "raba660670"),                                  ⑤
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Edges of the trough, the broader scope.
④ Trough of interest, a narrower scope to find the reflectance value.
⑤ Name of the index to be stored in the raster data.

5.5 Spectral ratios

Another popular and straightforward index is band ratios, where reflectance at wavelength X is divided by reflectance at wavelength Y.

```r
# Calculate ratio
ratio_570630 <- reflectance |>                                  ①
  HSItools::calculate_band_ratio(                               ②
    edges = c(570, 630),                                        ③
    ratio_name = "r570r630")                                    ④
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Wavelength X and Y, numerator and denominator.
④ Name of the index to be stored in the raster data.

5.6 Derivatives - ToDo

> **ToDo**
>
> This is a feature yet to be added.

Another popular and straightforward index is band ratios, where reflectance at wavelength X is divided by reflectance at wavelength Y.

```
# Calculate derivative
derivative_550 <- reflectance |>                                    ①
  HSItools::calculate_derivative(                                  ②
    band = 550,                                                    ③
    derivative_name = "der550")                                    ④
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Wavelength of choice.
④ Name of the index to be stored in the raster data.

5.7 Differences

```
# Calculate the difference
difference_650675 <- reflectance |>                                ①
  HSItools::calculate_band_difference(                             ②
    edges = c(650, 675),                                           ③
    difference_name = "dif650dif670")                              ④
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Wavelength X and Y.
④ Name of the index to be stored in the raster data.

5.8 Normalized difference index (NDI)

```
# # Calculate NDI
ndi_650675 <- reflectance |>                                       ①
  HSItools::calculate_ndi(                                         ②
    edges = c(650, 675),                                           ③
    ndi_name = "ndi650ndi670")                                     ④
```

① Reflectance data is either in memory or from a disk.
② Calculation function.
③ Wavelength X and Y.
④ Name of the index to be stored in the raster data.

5.9 Other

5.9.1 Red Edge Minimum Point (REMP) - ToDo

> **ToDo**
>
> This is a feature yet to be added.

This index was introduced in the paper "Ghanbari, H., Zilkey, D.R., Gregory-Eaves, I., Antoniades, D., 2023. A new index for the rapid generation of chlorophyll time series from hyperspectral imaging of sediment cores. Limnology and Oceanography: Methods 21, 703–717.

```
# Calculate  REMP
remp <- reflectance |>                                    ①
  HSItools::calculate_remp()                              ②
```

① Reflectance data is either in memory or from a disk.
② Calculation function.

6 Visualizing the data

6.1 RGB, NIC and CIR preview

Preview RGB with the scale.

```
# Preview RGB
p_rgb <- reflectance |>
  # Plot
  plot_raster_rgb(                                                    ①
    calibration = pixel_to_distance(hsi_tools_core),                  ②
    # extent = roi_sml[1, ])                                          ③

# Print
p_rgb
```

① Function to plot RGB.
② Call to HSItools core output of the Shiny app storing the depth calibration.
③ Optional call to use extent.

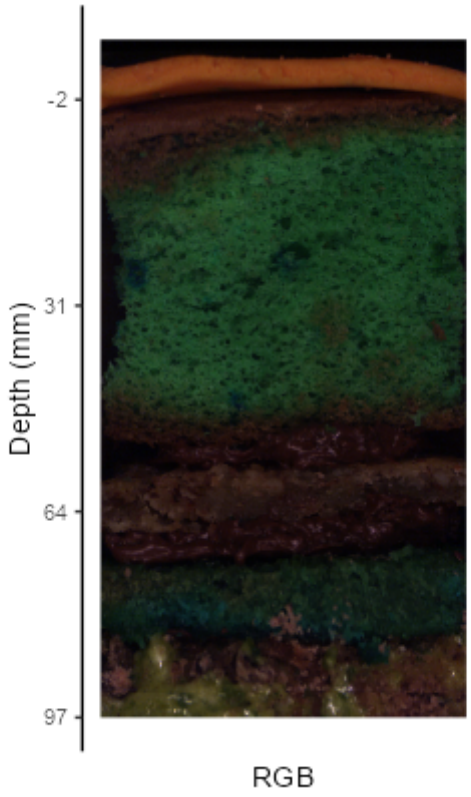


Figure 6.1: Plot of RGB data with depth scale.

6.2 Index map

For example, we can plot the spatial map of RABD660:670max.

```
# Plot RABD
p_rabd <- rabd_max |>
  plot_raster_proxy(                                                    ①
    hsi_index = names(rabd),                                            ②
    calibration = pixel_to_distance(hsi_tools_core),                    ③
    palette = "viridis",                                                ④
    # extent = roi_sml[1, ])                                            ⑤

# Print
p_rabd
```

① Function to plot index map.

② Name specification.
③ Call to HSItools core output of the Shiny app storing the depth calibration.
④ Palette of choice, one of the {viridis} options.
⑤ Optional call to use extent.

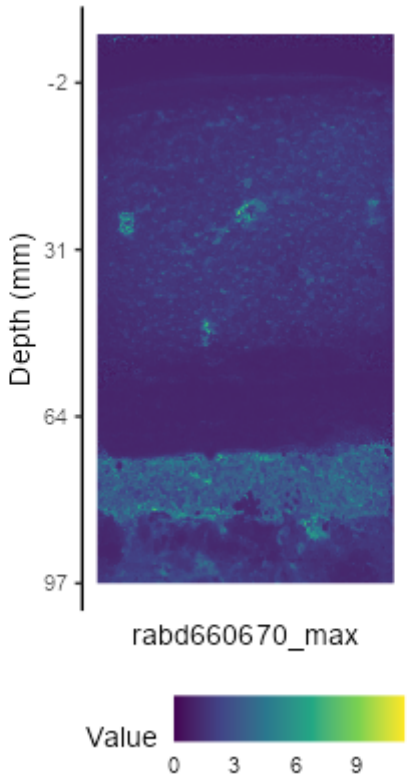


Figure 6.2: Plot of $RABD_{660:670max}$ data with depth scale.

6.3 Index plot

We can extract profile of the selected index using one of the predefined ROIs. We use the small, green ROI in the middle.

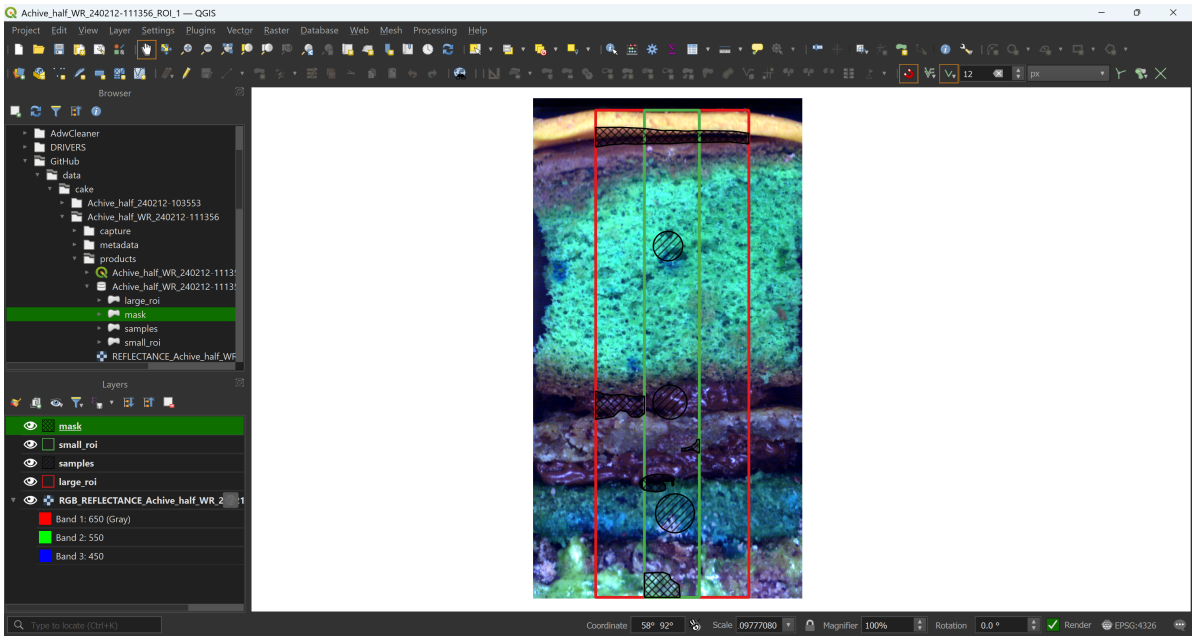


Figure 6.3: Preview of ROIs

```
# Plot series
p_rabd_series <- rabd |>
  # Plot
  HSItools::plot_profile_spectral_series(
    hsi_index = names(rabd),
    calibration = pixel_to_distance(hsi_tools_core),
    extent = rois_sml[1, ])

# Print
p_rabd_series
```

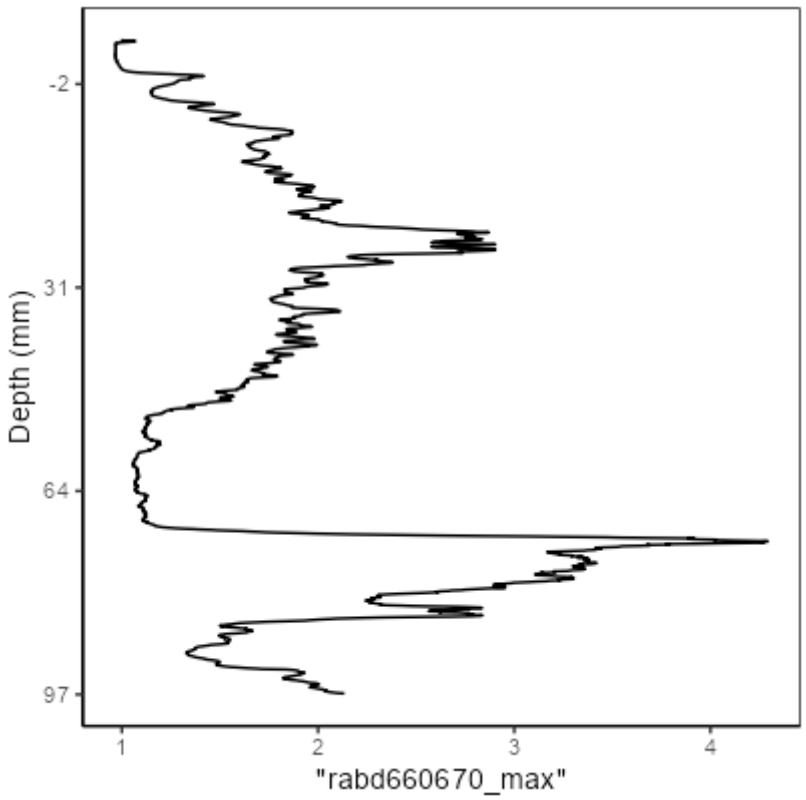


Figure 6.4: Profile of $\text{RABD}_{660:670\text{max}}$ using the small ROI.

We can display it side by side using the magic of {patchwork}.

```
# Plot
p_rgb + p_rabd + p_rabd_series
```

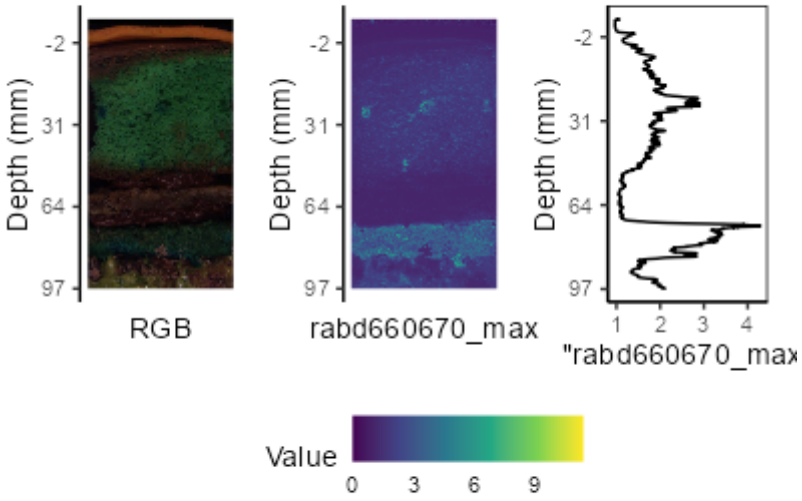


Figure 6.5: RGB, $\text{RABD}_{660:670\text{max}}$ map and $\text{RABD}_{660:670\text{max}}$ profile.

6.4 Spectrum plot

Finally, you might be interested in plotting the average spectrum from the selected area. We'll use the uppermost "sample" from the ROI preview.

```
# Plot reflectance profile
p_sample <- reflectance |>
  plot_profile_spectral_profile(extent = rois_smp[1, ])

# Print
p_sample
```

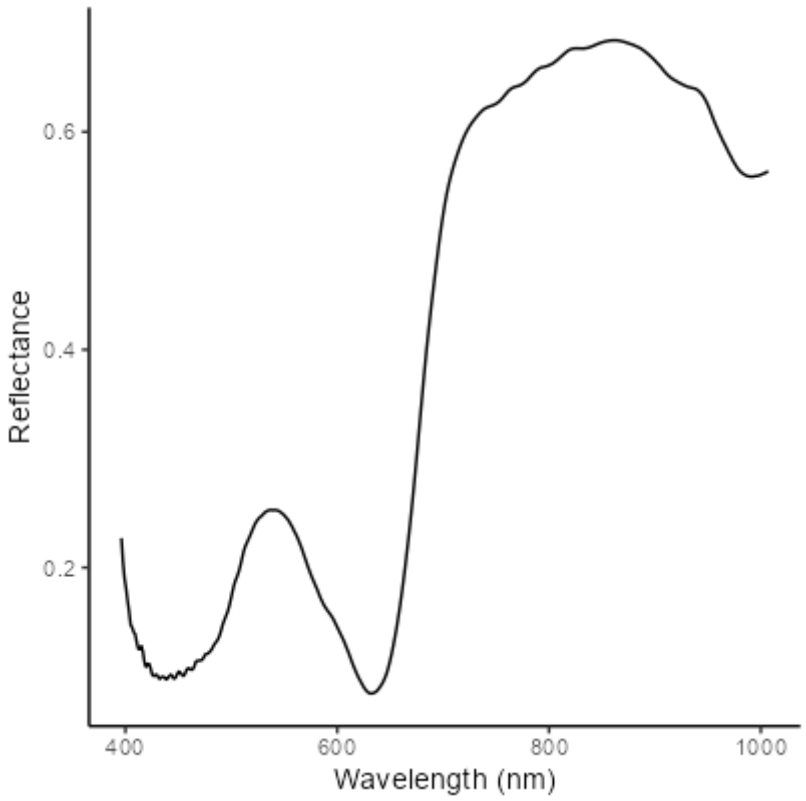


Figure 6.6: Reflectance profile of the topmost “sample” area.

7 Extracting the data

Once you're happy with your indices and plots, you can also extract the underlying data.

7.1 Index profile

You can extract the data that was used to plot the curve:

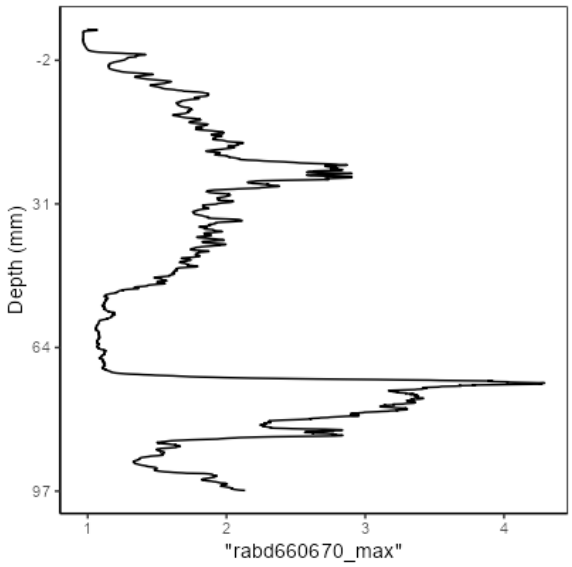


Figure 7.1: Profile of $\text{RABD}_{660:670\text{max}}$ using the small ROI.

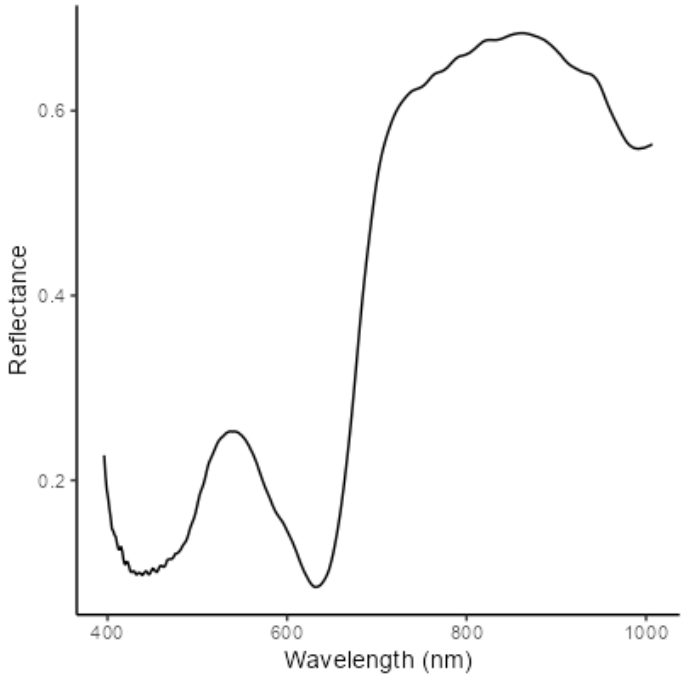
```
# Extract
d_rabd <- rabd |>
  extract_spectral_series(
    index = names(rabd),
    calibration = pixel_to_distance(hsi_tools_core),
    extent = rois_sml[1, ])

# Print
d_rabd
```

```
# A tibble: 1,606 × 5
      x     y rabd660670_max depth.mm tube.mm
  <dbl> <dbl>          <dbl>    <dbl>   <dbl>
1   657 1608.          0.994     51.9   -9.05
2   657 1606.          1.03      52.0   -8.98
3   657 1606.          1.07      52.1   -8.91
4   657 1604.          1.07      52.1   -8.85
5   657 1604.          1.04      52.2   -8.78
6   657 1602.          1.01      52.3   -8.71
7   657 1602.          0.982     52.3   -8.65
8   657 1600.          0.969     52.4   -8.58
9   657 1600.          0.967     52.5   -8.52
10  657 1598.          0.971     52.5   -8.45
# i 1,596 more rows
# i Use `print(n = ...)` to see more rows
```

Figure 7.2: Tibble of extracted values.

7.2 Spectral profile

Similarly, You can extract the data that was used to plot the curve:



```
d_reflectance <- reflectance |>
  # Extract profile
  extract_spectral_profile(extent = rois_sml[1, ])
```

```
# A tibble: 1 × 392
  `396.24` `397.68` `399.12` `400.57` `402.01` `403.45` `404.9` `406.35` `407.79` `409.24` `410.69` `412.15` `413.6` `415.05` `416.51` `417.96` `419.42`
     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1    0.191    0.175    0.163    0.156    0.146    0.137   0.128    0.126    0.125    0.121    0.115    0.111   0.112    0.113    0.109    0.102   0.0973
# ℹ 375 more variables: `420.88` <dbl>, `422.34` <dbl>, `423.8` <dbl>, `425.26` <dbl>, `426.72` <dbl>, `428.19` <dbl>, `429.65` <dbl>, `431.12` <dbl>,
#   `432.58` <dbl>, `434.05` <dbl>, `435.52` <dbl>, `436.99` <dbl>, `438.46` <dbl>, `439.93` <dbl>, `441.41` <dbl>, `442.88` <dbl>, `444.36` <dbl>,
#   `445.83` <dbl>, `447.31` <dbl>, `448.79` <dbl>, `450.27` <dbl>, `451.75` <dbl>, `453.23` <dbl>, `454.71` <dbl>, `456.2` <dbl>, `457.68` <dbl>,
#   `459.17` <dbl>, `460.65` <dbl>, `462.14` <dbl>, `463.63` <dbl>, `465.12` <dbl>, `466.61` <dbl>, `468.1` <dbl>, `469.59` <dbl>, `471.09` <dbl>,
#   `472.58` <dbl>, `474.08` <dbl>, `475.57` <dbl>, `477.07` <dbl>, `478.57` <dbl>, `480.07` <dbl>, `481.57` <dbl>, `483.07` <dbl>, `484.57` <dbl>,
#   `486.07` <dbl>, `487.58` <dbl>, `489.08` <dbl>, `490.59` <dbl>, `492.09` <dbl>, `493.6` <dbl>, `495.11` <dbl>, `496.62` <dbl>, `498.13` <dbl>,
#   `499.64` <dbl>, `501.15` <dbl>, `502.66` <dbl>, `504.18` <dbl>, `505.69` <dbl>, `507.21` <dbl>, `508.72` <dbl>, `510.24` <dbl>, `511.76` <dbl>, …
# ℹ Use `colnames()` to see all variable names
```

Or if you'd like to get the index value, for example to use for calibration

```
d_rabd_value <- rabd |>
  extract_spectral_profile(extent = rois_sml[1, ])
```

```
# A tibble: 1 × 1
  rabd660670_max
           <dbl>
1           1.79
```

8 Summary

In summary, this book has just begun.

References