945358-BIGPICTURE
Central Repository for Digital Pathology
WP4 - Tools for accessing, annotating and mining digital slides
D4.04 - WSI format to DICOM conversion tools

Lead contributor	29 - SECTRA AB
	erik.o.gabrielsson@sectra.com

Document History

Version	Date	Description
V1.1	21 Dec 2022	First Draft
V1.2	22 Dec 2022	Comments
V1.3	10 Jan 2023	Second Draft
V1.4	25 Jan 2023	Final Version

Abstract

To provide a uniform format for the Whole Slide Images (WSIs) to be included in the BigPicture repository it has been decided that all images should be stored in the vendor neutral DICOM format. However, most nodes use scanners that produce and/or has large archives of WSIs in the scanner manufactures' proprietary format. To enable submission of such images to the BigPicture repository a converter has been developed, capable of reading WSIs in common proprietary formats and converting them to DICOM. This report describes the development of the converter, how it can be used in a BigPicture submission workflow and compares it to other available WSI converters.

Methods

Software and hardware used

Tests were run on a computer equipped with an Intel Core i7-10510U processor (4 cores, 8 threads, 8 MB cache), 16 GB of DD4-3200 RAM, and 512 GB PCIe Gen3 x4 NVMe SSD storage.

The collection of WSI to DICOM converters tested is detailed in Table 1, and other software used during testing in Table 2. Tests were run in Ubuntu with Windows Subsystem for Linux 2 (WSL2) running in a Windows 10 host. During benchmarking no other applications were active and the computer otherwise left idle. WSL2 was limited to using 8 GB of memory.

Table 1. WSI to Dicom converter software used during testing in this report.

Name	Version	URL	License	Installation
WsiDicomizer	0.3.1	https://github.com/imi-bigpicture/wsidicomizer	Apache 2.0	pip install wsidicomizer
wsi2dcm	1.0.3	https://github.com/GoogleCloudPlatform/wsi-to-dicom-converter	Apache 2.0	apt-get install wsi2dcm
wsi2dcm-dev		https://github.com/GoogleCloudPlatform/wsi-to-dicom-converter/tree/ef839703d6cf3a6e50a5204924c100aa03420c00	Apache 2.0	Git clone and built according to instructions
OrthancWSIDicomizer	1.1	https://www.orthanc-server.com/static.php?page=wsi	AGPL	apt-get install orthanc-wsi

Table 2. Other software used during testing in this report.

Name	Version	URL	License	Installation
WsiDicom	0.4.0	https://github.com/imi-bigpicture/wsidicom	Apache 2.0	pip install wsidicom
OpenSlide Python	1.2.0	https://openslide.org/api/python/	LGPL 2.1	pip install openslide-

				python
libjpeg-turbo	2.1.2	https://libjpeg-turbo.org/	BSD-style	apt-get install libjpeg-turbo
OpenSlide	3.4.1	https://openslide.org/	LGPL 2.1	apt-get install openslide-tools
Hyperfine	1.15.0	https://github.com/sharkdp/hyperfine	Apache 2.0, MIT	apt-get install hyperfine
Ubuntu	22.04.1	https://ubuntu.com	Mix	Installed from Microsoft Store

Images used for testing

Table 3 describes the images used during testing. All images are scans of the same slide at 20X or 40x magnification and are licensed as Creative Commons designation CC0 1.0 Universal (CC0 1.0) Public Domain Dedication.

Table 3. Image files used for testing in this report.

Format	**Filename**	**Size (MB)**	**Pixels**	**URL**
Aperio svs	CMU-1.svs	173	1.5 GP	https://openslide.cs.cmu.edu/download/openslide-testdata/Aperio/CMU-1.svs
Hamamatusu ndpi	CMU-1.ndpi	193	2.0 GP	https://openslide.cs.cmu.edu/download/openslide-testdata/Hamamatsu/CMU-1.ndpi
3DHistech mrxs	CMU-1.mrxs	538	6.5 GP	https://openslide.cs.cmu.edu/download/openslide-testdata/Mirax/CMU-1.zip
Trestle tiff	CMU-1.tif	158	1.1 GP	https://openslide.cs.cmu.edu/download/openslide-testdata/Trestle/CMU-1.zip

Testing of conversion loss

Changes in image data during conversion to DICOM WSI, i.e. conversion loss, were tested by converting representative images of different formats (see Table 3) into DICOM WSI, opening the produced files with WsiDicom, reading representative regions of image data from the files and comparing the read image data with the image data read for the same region from the original image file using OpenSlide-Python. When re-compression was used the image was re-compressed in Jpeg format with a quality setting of 80 and 420 subsampling. To quantify the loss, the difference between the converted and original image data was calculated as a pixel-by-pixel difference and the root mean square error calculated per color component. To visualize the loss the difference between the converted and original image increased in contrast by 20.

Testing of conversion time

The benchmarking tool hyperfine was used to time the conversion time of WSI files listed in Table 3 using the command-line interface of the respective converters listed in Table 1. To avoid limitations in how pyramid reconstruction was handled only the base layers were converted. A tile size of 256x256 pixels was used, allowing for lossless conversion of the Aperio svs file (with matching native tile size) if implemented in the converter. For re-encoding Jpeg compression with quality setting 80 was used. The converters were allowed to use 8 threads. Hyperfine was configured to first run 2 warmup runs followed by 5 timed runs. The file system cache was synced and cleared between each run.

Testing of loading time

WsiDicom was used to load DICOM WSIs converted using different converters and conversion options, as outlined in Table 4, and read a single tile from the base layer. The same source image, CMU-1.svs, was used. After 10 warmup rounds the measurement was repeated 10 times. The tile organization and the type of table offset (if present) were read out from the converted files using WsiDicom.

Table 4. Converters and settings used for testing loading times of converted WSI DICOM files.

Converter	Setting
WsiDicomizer	With basic table offset (BOT)
WsiDicomizer	With extended table offset (EOT)
WsiDicomizer	Without table offset
OrthancWSIDicomizer	Default
wsi2dcm	Default
wsi2dcm-dev	Default

Results

Summary and overview of DICOM conversion tools implementation

The DICOM conversion tools for BigPicture usage involves four software projects:

- WsiDicom for handling DICOM WSI files. WsiDicom was created as part of the earlier deliverable *D4.03 Report on unified open digital slide and annotation format specification*[1] as a tool for reading DICOM WSI files. For this deliverable the package was extended to also

[1] Available from https://cordis.europa.eu/project/id/945358/results/es

write DICOM WSI files. To a large extent the tool uses the open source DICOM library pydicom for handling the DICOM format.[2]

- OpenTile[3] for reading image data and image data from compatible TIFF-based WSI files using the open-source package tifffile.[4] Compared to tifffile the tool provides a higher-level interface to access data, for example providing methods to get tiled image data by tile coordinate.
- WsiDicomizer, an extension of WsiDicom for handling image data and metadata from non-DICOM files. WsiDicomizer uses tools such as OpenSlide and OpenTile to read other file formats and is design to allow simple extension to support other file formats. WsiDicomizer can either be called from other Python code or from a command prompt using a command-line interface.
- BigPicture converter, an example on how to integrate the WSI DICOM converter into a submission workflow together with BigPicture metadata.

Table 5 summarizes the functionality of the presented conversion tools.

To encourage community contribution, all projects except the BigPicture converter have been released under a permissive open-source license (Apache Licence 2.0[5]) and published at BigPicture's public GitHub organization.[6] The BigPicture converter will likely also be released as open-source once the BigPicture metadata format and associated tools are released. To lower the threshold for contributing we have focused on code readability, using Python,[7] a programming language emphasizing code readability, type hints for input arguments and return types,[8] and descriptive docstrings.[9]

WsiDicomizer implements format converters as subclasses of WsiDicom, allowing the opened file to be used as if it would be in DICOM format. The *open()*-method has been overridden in order to handle non-DICOM-files, and a *convert()*-method has been added to convert the opened image to DICOM. To enable reading of other WSI formats several image readers are implemented in WsiDicomizer, allowing WsiDicomizer to arrange the image data in the opened file into pyramid levels, read out relevant metadata, and provide access to the image data.

One of the implemented image readers is based on OpenSlide, allowing many common WSI formats to be converted. OpenTile is used for another image reader, offering more advanced support for certain WSI formats.

---

[2] https://pydicom.github.io/

[3] https://github.com/imi-bigpicture/opentile

[4] https://github.com/cgohlke/tifffile

[5] https://www.apache.org/licenses/LICENSE-2.0

[6] https://github.com/imi-bigpicture

[7] https://www.python.org/

[8] https://peps.python.org/pep-0484/

[9] https://peps.python.org/pep-0257/

The *convert()*-method uses the DICOM WSI writer implemented in WsiDicom to level-by-level convert the opened file to a DICOM WSI file. File metadata are, when possible, parsed into DICOM metadata and included in the converted files.

Due to the tiled arrangement of WSIs and the mix of IO-operations (reading and writing image data) and CPU-operations (processing image data), conversion of WSIs into a new format can in general benefit from dividing the conversion into multiple workloads than can run in parallel, i.e. threading. The conversion tool allows the conversion process to be distributed into multiple threads and the image readers are designed to allow multiple conversion threads to be able to executed in parallel.

To allow for additional metadata besides parsable metadata from the input image file, WsiDicomizer can provide user-defined metadata in the form of DICOM datasets to the converter. These datasets will then be merged with other metadata into the DICOM WSI files. In the BigPicture converter this is used to populate required DICOM metadata with BigPicture metadata, such as a slide identifier. The BigPicture converter also uses metadata from the converted DICOM files to populate the BigPicture metadata, such as image size and pixel spacing.

Table 5. Summary of converter functionality.

Language	Python
License	Apache 2.0
Format	
Aperio SVS	Lossless
Hamamatsu NDPI	Lossless
Philips TIFF	Lossless
3DHistech TIFF	Lossless
Leica SCN	Lossy
3DHistech MRXS	Lossy
Trestle TIF	Lossy
Ventana BIF	Lossy
Zeiss CZI	Lossy
DICOM*	Lossless
Tile arrangement	
Reading	Full- or sparse
Writing	Full
Tile index	Basic, extended, or no offset table

Multiple focal planes	Yes
Multiple optical paths	Yes

* DICOM can be converted to DICOM formatted according to BigPicture specifications.

DICOM WSI data arrangement

The data arrangement used in WsiDicom is compatible with DICOM WSI and illustrated in Figure 1. The access to image data is abstracted in the *ImageData* class, where WsiDicom uses an implementation for supporting DICOM files. WsiDicomizer uses format specific *ImageData* implementations to support other formats, providing a way to use other WSI formats in the same way as DICOM.

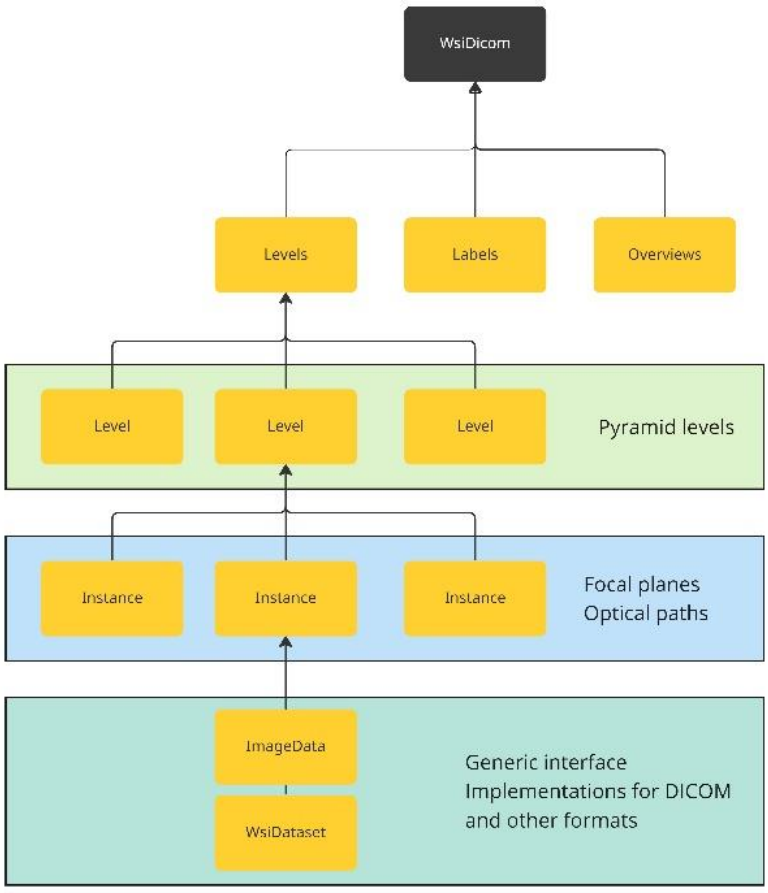

Figure 1. Data arrangement in WsiDicom. WsiDicom contains collections of level, label, and overview images (only levels shown further for simplicity). The levels collection contains the pyramid levels of the image. Each level can contain multiple instances, where it is possible that the instances represent different focal planes and/or optical paths. An instance combines image data (ImageData) and image metadata (WsiDataset).

DICOM WSI writer

The specification for DICOM WSI offers several options on how to represent image data[10] and producing a converter tool that implements all available options would be very challenging. As outlined in the deliverable D4.03 regarding best practices for writing DICOM files for fast viewing performance, BigPicture has decided that DICOM WSI files used within BigPicture should only use a subset of the options available. In summary, these options include:

- Image data should be fully tiled as opposed to sparsely tiled, i.e. all tiles of the image should be present in the image data.
- The image data should be indexed either using a basic offset table (BOT) or extended offset table (EOT).
- Image data should preferably be encoded in Jpeg, with the option to use Jpeg 2000 if needed.
- Tiles should have a square size and be of the same size across the image pyramid.
- The image pyramid should be dyadic, i.e. the scale between two neighbouring levels should be 2.
- A single file should be used per pyramid level, including all focal paths and if possible optical paths. If required due to different photometric interpretations, the pyramid level can be split into multiple files per optical path.

The low-level Python DICOM library pydicom was used to implement a DICOM WSI file writer. The file writing process is divided into the following steps:

1. Writing file preamble and file metadata dataset. These steps write the required DICOM file preamble followed by the file metadata dataset specifying the used transfer syntax (based on used compression), the instance UID, and the class UID for DICOM WSI.
2. Writing the metadata dataset.
3. Reserving space for BOT or EOT if used
4. Writing image data and creating an index for the BOT/EOT.
5. Writing the BOT or EOT if used.

Rewriting DICOM files

The DICOM WSI writer implemented in WsiDicom can be used to re-write any DICOM WSI files into the BigPicture specified format by using the *save()*-method. For sparse tiled files this includes converting the file to full tiled by generation of blank tiles. No re-compression is needed as the source file should use an image format for the tiles already compatible with DICOM (typically Jpeg or Jpeg2000). By default, a basic offset table is added to the files, with the option to instead use extended offset table or no table. However, using no table is not recommended.

For DICOM WSIs lacking pyramid levels, the *construct_pyramid()*-method can be used to create missing dyad pyramid levels by downscaling from the closest lower level.

Implementing thread-safe tile writing

The chosen tiling method chosen for DICOM WSIs in BigPicture, e.g. fully tiled, requires that tiles

[10] https://dicom.nema.org/dicom/dicomwsi/

are present in a specific order in the file, e.g. first by row from left to right, followed by z and then optical path. As the file position for where to write a tile depends on the total length of all prior tiles (in the order specified by the format) it is not possible to write a tile to file prior to the previous tiles has been written, i.e. tiles has to be sequentially written to file in the order specified by the format.

Sequential writing is implemented using a thread pool executor.[11] A thread pool executor can receive a sequence of units of work, such as tile coordinates to process, send (map) the unit to one of its treads, and keep track of the results from each tread so that the results are returned in the original order. The number of threads to use is user configurable and defaults to the number of CPUs available if not defined. To enable compatibility with libraries that cannot be threaded, specifying that a single worker should be used disables the thread pool.

The image that is to be written is divided into non-overlapping chunks containing sequential tile coordinates. The image data implementation can suggest a chunk size to use for efficient file processing, otherwise a default chunk size of 16 tiles is used. The thread pool executor sends the chunks to threads that read out image data for the tiles in the chunk and collects the results. The result from each thread, in the form of a sequence of bytes representing the image data for each chunk, is then written sequentially to file. Each tile represents one item in the image data and is wrapped in an item tag including the length of the tile prior to write.

Offset tables

To enable fast load time of images the DICOM WSI files used in BigPicture should have an offset table, a list containing the byte position of each tile (or specifically frame) in the image. Without such a table the position of each tile must be determined by parsing the image data, which can result in several gigabytes of data being searched.

DICOM specifies two types of offset tables, basic offset table (BOT) and extended offset table (EOT). Their main differences are:

- BOT uses 32-bit unsigned integers to specify frame positions, while EOT uses 64-bit unsinged integers. This limits the usage of BOTs to files with image data less then about 4.3 GB, while EOT is in practice limitless.
- BOT only specifies frame positions, while EOT stores frame positions and frame lengths
- BOT is stored as the first item in the PixelData element, while the EOT is stored in two separate elements, Extended Offset Table and Extended Offset Table Lengths. The EOT elements are typically present immediately before the ImageData element.

The DICOM WSI writer implements support for both basic and extended offset tables. The option to write BOT, EOT, or neither, is user selectable.

As BOT and EOT should be present before the image data and contain data that is only available after the image data has been written to file (the location of each tile), space is reserved for BOT/EOT prior to writing the image data. The size needed for both BOT and EOT is calculated from the number of tiles to write and the size of each record in the table (e.g. either 32 or 64 bits).

During writing of image tiles to the file, the start position of each tile (including its *item*

[11] https://docs.python.org/3/library/concurrent.futures.html

encapsulation) is recorded, producing a list of tile positions in relation to the start of the file. After all tiles has been written, the positions are re-calculated to represent offsets in relation to the first tile in the image data and tile length (for EOT). These lists are then written as 32- or 64-bits integers at the reserved BOT/EOT space.

Additionally, the DICOM WSI reader implemented in WsiDicom supports both reading BOT and EOT as well as files without offset table.

While EOT are technically more suited for WSIs, as it is not uncommon for base levels to exceed the limitation of BOT, support for reading EOT is not yet as widespread as for BOT. The default option is thus to write a BOT, with the option to use EOT or no offset table (not recommended) if needed. Unfortunately, it is difficult to predict the final image data size, especially if the image needs to be re-compressed. If the final image data exceeds the capability of the BOT an exception will be raised, informing the user that an EOT should be used instead. A better option would be to enable automatic selection of the table type based on image data size or to re-write the file with an EOT should the BOT overflow.[12]

Image readers

WsiDicomizer uses image readers to access image data from different image formats. As an image reader based on OpenSlide is implemented, the converter can be used together with any of the many format that OpenSlide supports. The details of the conversion process using the OpenSlide image reader is presented in the section Generic conversion using OpenSlide. The OpenSlide image reader has some limitations, primarily in that it can only be used for re-encoding and only single focal planes and a single optical path is supported. Commonly used formats that can technically be converted without re-encoding, i.e. lossless, include the TIFF-based formats Aperio SVS and Hamamatsu NDPI. To offer lossless conversion of certain TIFF-based formats an image reader based on OpenTile is used, see the section Lossless conversion of TIFF-based formats. An additional image reader supports conversion of Zeiss CZI files, see the section Conversion of Zeiss CZI.

Implementing support for an image format is relatively simple if there exist tools and libraries to read the image format.

Lossless conversion of TIFF-based formats

Image data can in general be converted to DICOM WSI format if the original image data stores the image in non-overlapping tiles (see Figure 2) and the compression is allowed according to the DICOM standard for WSI. Some formats that (typically) fulfill these requirements and that additionally are TIFF-based are:

- Aperio SVS
- Hamamatsu NDPI
- Philips TIFF
- OME-TIFF
- 3DHistech TIFF

[12] https://github.com/imi-bigpicture/wsidicom/issues/76

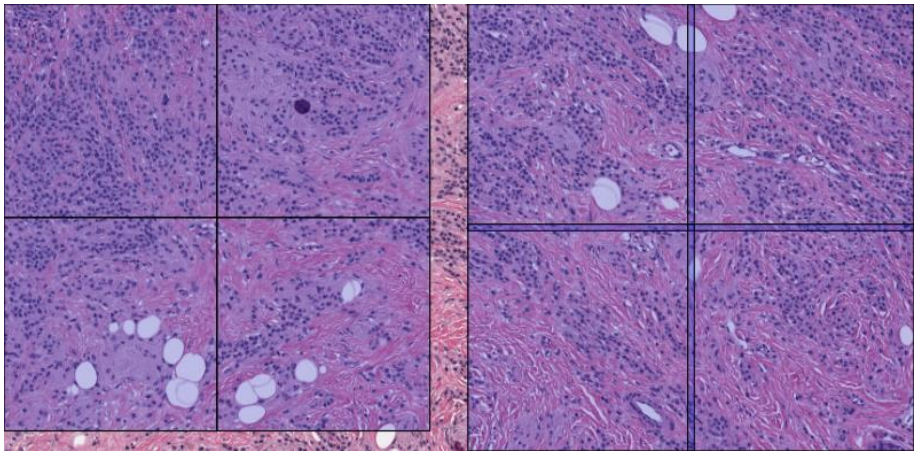

Figure 2. Example of tile arrangement without overlap (left) and with overlap (right).

An image reader was implemented for TIFF-based files with non-overlapping image data. Although these files are TIFF-based, they often extend the TIFF-format using proprietary tags and similar. To read image data and metadata from TIFF-files the Python library tifffile was used, as it implements most of the important format specifics and provides simple methods for reading image data and either processed or raw metadata.

To further simplify the image data and metadata reading from these files, the higher-level Python library OpenTile was created. OpenTile uses tifffile to access the image data and metadata, and adds methods access the image data by pyramid level and tile position as well presenting uniform processed metadata.

Philips TIFF

Philips TIFF-files can be sparsely tiled, which is detected as a zero-length frame length for the tile. To provide blank-tiles for missing tiles as similar as possible to the present tiles, a method to replace the compressed data in a Jpeg-frame with uniform grayscale data while keeping the same Jpeg-headers was implemented.

Philips TIFF-files contains metadata both in the regular TIFF-tags but also as a DICOM dataset in xml-format in the *description*-tag. Metadata is parsed from both the regular tags and the included DICOM dataset.

Aperio SVS

During testing it was observed that a small number of Aperio SVS files had corrupted tiles, present at the right and/or bottom edge of scaled levels (i.e. not the base level). The corrupted tiles were detectable by a zero-length frame length. To prevent corrupted tile to be converted, a method to flag corrupted edges was implemented. When a tile along a corrupted edge is accessed, a down-scaled tile from a lower pyramid level is created, cached, and return instead. The tile is compressed using the same compression method as other tiles.

Metadata is available in and parsed from regular TIFF-tags and in a dictionary-like string in the *description*-tag.

Hamamatsu NDPI

In the Hamamatsu NDPI-format image data is stored in non-overlapping stripes instead of tiles. It is thus not possible to directly arrange the image data into tiles preferred for DICOM WSI. However, as the image data is (typically) Jpeg compressed, another method, based on stacking stripes and lossless cropping them into tiles, can be used.

In Jpeg compression the image is divided into a grid of so called minimum coded units (MCUs). Depending on the used subsampling, these MCUs are either 8 or 16 pixels in width and height. During compression each MCU is first converted to a frequency-domain representation (one DC component and several AC components) using a discrete cosine transform. The components are then quantized according to a provided quantization table. This step is irreversible, making Jpeg compression lossy. Finally, the quantized components are run length encoded using Huffman coding. The Huffman coding is lossless, thus enabling decoding, manipulation, and re-coding of the data without introducing additional compression artifacts. Here, the DC components is encoded as the delta from the previous MCUs DC component, i.e. only the first MCU in a row has the absolute DC value encoded. Optionally the DC value can be reset after a defined number of MCUs.

Except for the DC component (which is encoded as the difference to the previous MCUs DC component), each MCU is independent from its surrounding MCUs. If the previous MCU is changed, one simply must calculate a new DC component delta for the current MCU based on the new previous MCU. As the Huffman coding is lossless, one can change just the DC offset without introducing other changes to the data and thus operate on an image to crop or join discrete MCUs by only doing lossless updates of the DC delta.

Specifically, for NDPI-files, the base layer typically has a stripe size of 4096x8 pixels and the DC delta is reset after each stripe. To for example produce tiles of size 512x512 pixels, as illustrated in Figure 3, one can:

1. Vertically join 64 stripes, producing a frame of 4096x512 pixels. As the DC delta is reset after each stripe no changes to the DC components are needed.
2. Crop out 8 tiles of 512x512 from coordinates (0, 0), (512, 0), etc. For each row in the frame the actual DC value is calculated at the boundary to a new tile, so that an absolute DC value can be encoded into the first MCU of the next tile.

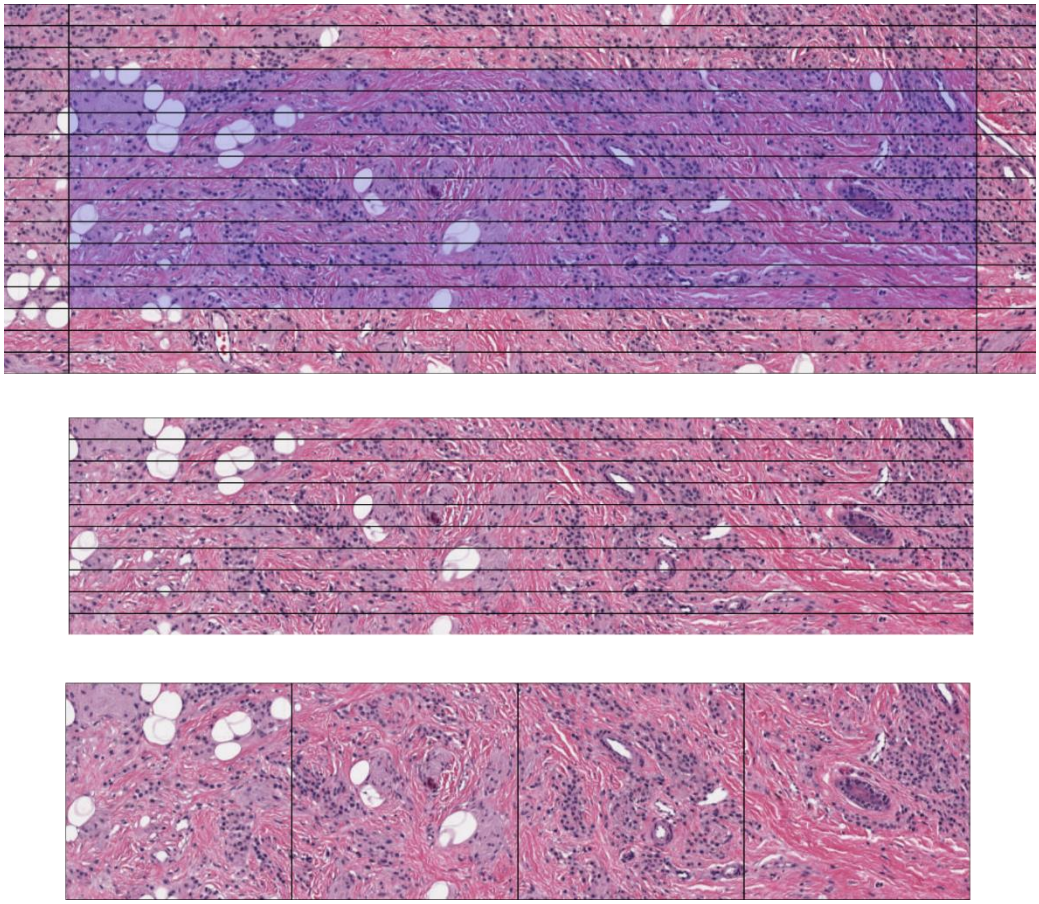

Figure 3. Concatenating stripes (top) to form a larger frame (middle) to crop out tiles from (bottom). The number of stripes to concatenate is reduced for clarity.

Metadata is available and parsed from regular TIFF-tag and a selection of “private” TIFF-tags with known content.

Generic conversion using OpenSlide

OpenSlide offers support for several WSI formats in a uniform way, including:

- Aperio SVS
- MIRAX MRXS
- Trestle TIF
- Ventana BIF
- Hamamatsu NDPI, VMS, VMU
- Sakura SVSLIDE

OpenSlide handles both reading and decompression of image data and stitching of tiles or regions into complete images in RGBA-format. The image reader based on OpenSlide uses the *read_region()*-method to provide tiled and compressed image data to the DICOM WSI writer.

When requesting a tile from a pyramid level, the corresponding pixel region in the level is calculated based on the selected tile size. The image data for the tile is then read out using OpenSlide into a raw Numpy array.

To avoid processing image data representing blank tiles, the image data is first check to see if the array is filled with zeros (i.e. blank). If the data is blank, a pre-generated blank tile is returned.

Otherwise, the data is further processed before compression. Importantly the obtained data is in ARGB-format with pre-multiplied alpha, i.e., the alpha component has been applied to red, green, and blue component. This is converted to non-pre-multiplied RGBA using a C-method provided in the OpenSlide-Python library. Finally, a Pillow Image is created from the data and the alpha-component dropped. The resulting RGB-Pillow Image is then encoded using the configured compression method.

Currently OpenSlide only support reading of a single focal plane and optical path.

Conversion of Zeiss CZI

Zeiss CZI WSI files are supported by an implemented image reader in WsiDicomizer. In a CZI file image data is stored in overlapping blocks with arbitrary overlap. The image reader is based on the open-source library CziFile[13], which provides access to the image blocks and convenient ways to decompress the image data into Numpy arrays.

To read an arbitrary region, such as the region representing a tile to be converted, several blocks might need to be accessed, decompressed, and stitched together. A very simple stitching mechanism is implemented with no correlation or blending between parts to be stitched. As the overlap is arbitrary the stitching can't be done lossless; decompression and recompression is needed.

CziFile has no support for handling pyramid levels and thus only the base level can be converted. The pyramid can later be created using the *construct_pyramid()*-method in WsiDicom.

Since implementing the support for Zeiss CZI files using CziFile, Zeiss has released the library libCZI[14] and the python wrapper pylibCZIrw[15] for reading and writing CZI files under the more permissive LGPL-3.0-licencse. A re-write was done to use these libraries for reading CZI-files, but the result did not provide better performance.

Re-compression

Whenever re-compression of image data is needed WsiDicomizer can be configured to use either Jpeg or Jpeg2000 with selectable compression quality and, for Jpeg, subsampling.

Image reader threading support

The conversion tool allows the conversion process to be distributed into multiple threads. To circumvent the limitations of the Global interpreter lock (GIL), that prevents multiple threads to execute Python code simultaneously, the conversion tool uses methods that can release the GIL whenever possible, for example:

[13] https://github.com/cgohlke/czifile/

[14] https://github.com/ZEISS/libczi

[15] https://pypi.org/project/pylibCZIrw

- When processing stripes in Hamamatsu NDPI-files into tiles, the calls to the C library libjpeg-turbo (through the Python wrapper PyTurboJPEG[16]) releases the GIL.
- For formats implemented with the OpenSlide-based image reader, OpenSlide releases the GIL when retrieving image data (as OpenSlide is a C library). The image data is then further processed using Numpy, a library that also releases the GIL when possible.
- For the Zeiss CZI image reader implementation, the used CziFile Python-library uses calls to C-libraries and Numpy to decompress and process the data.
- GIL-releasing libraries are used for all other image processing operations (compression, decompression, scaling), such as Pillow[17], imagecodecs[18], and libjpeg-turbo.

Additionally, several built-in Python functions, for example for byte handling are implemented in C allowing calls to such functions to release the GIL.

To ensure that the image reader is thread-safe, the image reader implementations use a threading lock for file access whenever needed.

Integration of converter into extraction workflow

To demonstrate the possibility to integrate the DICOM converter into a dataset extraction workflow at a BigPicture node, where non-DICOM images must be converted and integrated with metadata according to the BigPicture metadata standard, the *BigPicture converter* example project was created and published internally on BigPicture's GitHub organization.

The tool defines two interfaces, one for retrieving BigPicture metadata for biological beings to export and of for retrieving WSI related to those biological beings. To use the tool, one must provide implementations of these interfaces, e.g. a method for getting metadata from LIMS and processing it into BigPicture metadata and a method for getting WSIs from a PACS. Using these two interfaces, the tool converts the related WSI to DICOM, encrypts the images, links the image metadata to the other metadata, and converters the metadata to submittable xml-files. The output is a folder with metadata and images formatted for upload to the BigPicture repository.

The project was used to prepare pilot datasets at the skin node and the cancer node.

As the project depends on the not-yet public *BigPicture metadata interface* project for handling metadata, the project is unfortunately not yet licensed as open source and publicly available.

Comparison to existing converters

Several other free tools exist for converting WSIs to DICOM. The two most popular alternatives are probably Google Cloud Platform's wsi2dcm and Orthanc's OrthancWSIDicomizer, both which are used to compare the performance of the presented converter with. Other converters that were not

[16] https://github.com/lilohuang/PyTurboJPEG

[17] https://python-pillow.org/

[18] https://github.com/cgohlke/imagecodecs

considered include Bio-Formats' bfconvert tool (restrictive licence, limited support for DICOM WSI)[19], Pathomation's PMA.start (not open-source)[20], dicom-wsi (not in active development).[21]

Conversion loss

The conversion loss due to possible re-encoding when converting WSI file formats into DICOM WSI using the tested converters are presented in Figure 4 and Figure 5. During lossless conversion, the conversion should introduce no noise into the converted image, and the difference between the original and converted image should be zero. If the conversion involves re-encoding, and if the used compression method is not lossless (e.g. if Jpeg compression is used), the conversion will introduce additional noise into the image data.

Of the file formats tested, Aperio SVS is best supported for lossless conversion, as only the stable release of wsi2dcm uses re-encoding. For the Mirax and Trestle formats none of the tested converters supports lossless conversion, as they all use OpenSlide to read out the image data. For Hamamatsu NDPI, only WsiDicomizer supports lossless conversion.

It should be noted that the magnitude of the introduced noise is dependent on the compression methods used, both in the original file and in the converted file. Thus, the presented data should not be used to evaluate the conversion loss using other criteria than *lossless* or *lossy*.

[19] https://docs.openmicroscopy.org/bio-formats/6.11.1/users/comlinetools/conversion.html

[20] https://www.pathomation.com/pma-start/

[21] https://github.com/Steven-N-Hart/dicom_wsi

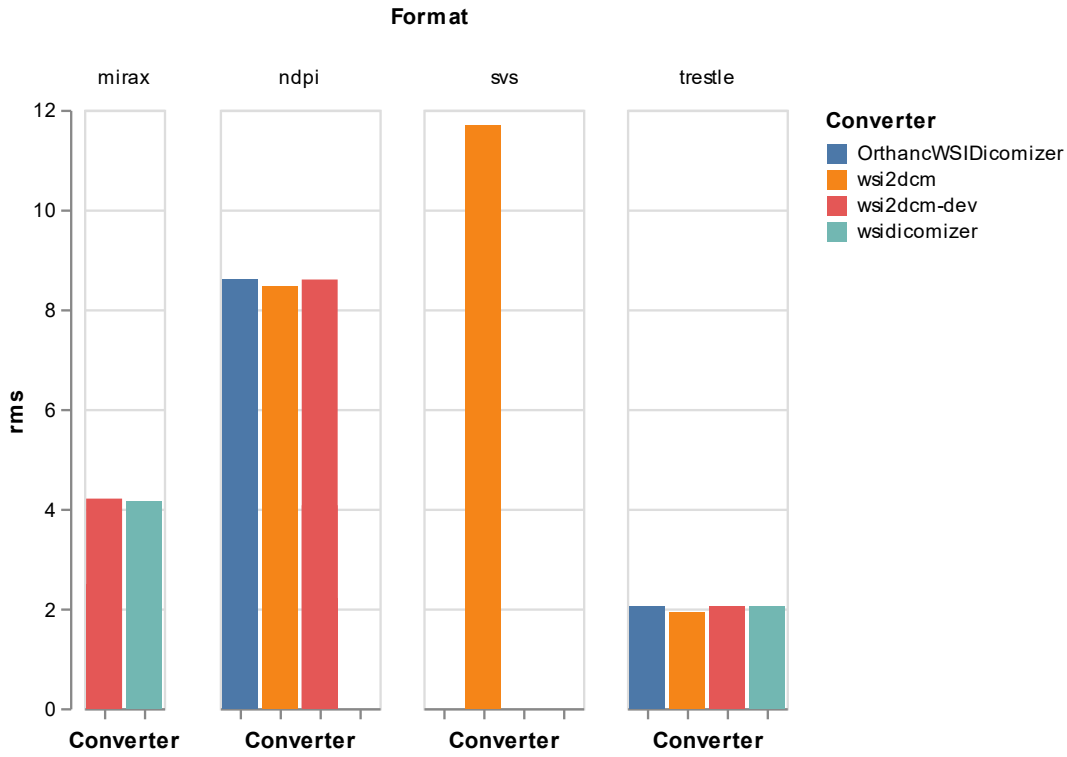

Figure 4. Conversion loss calculated as introduced noise after conversion for a sampled 200 x 200 pixel image region. Note that the magnitude of the conversion loss is dependent on the original compression and re-compression settings.

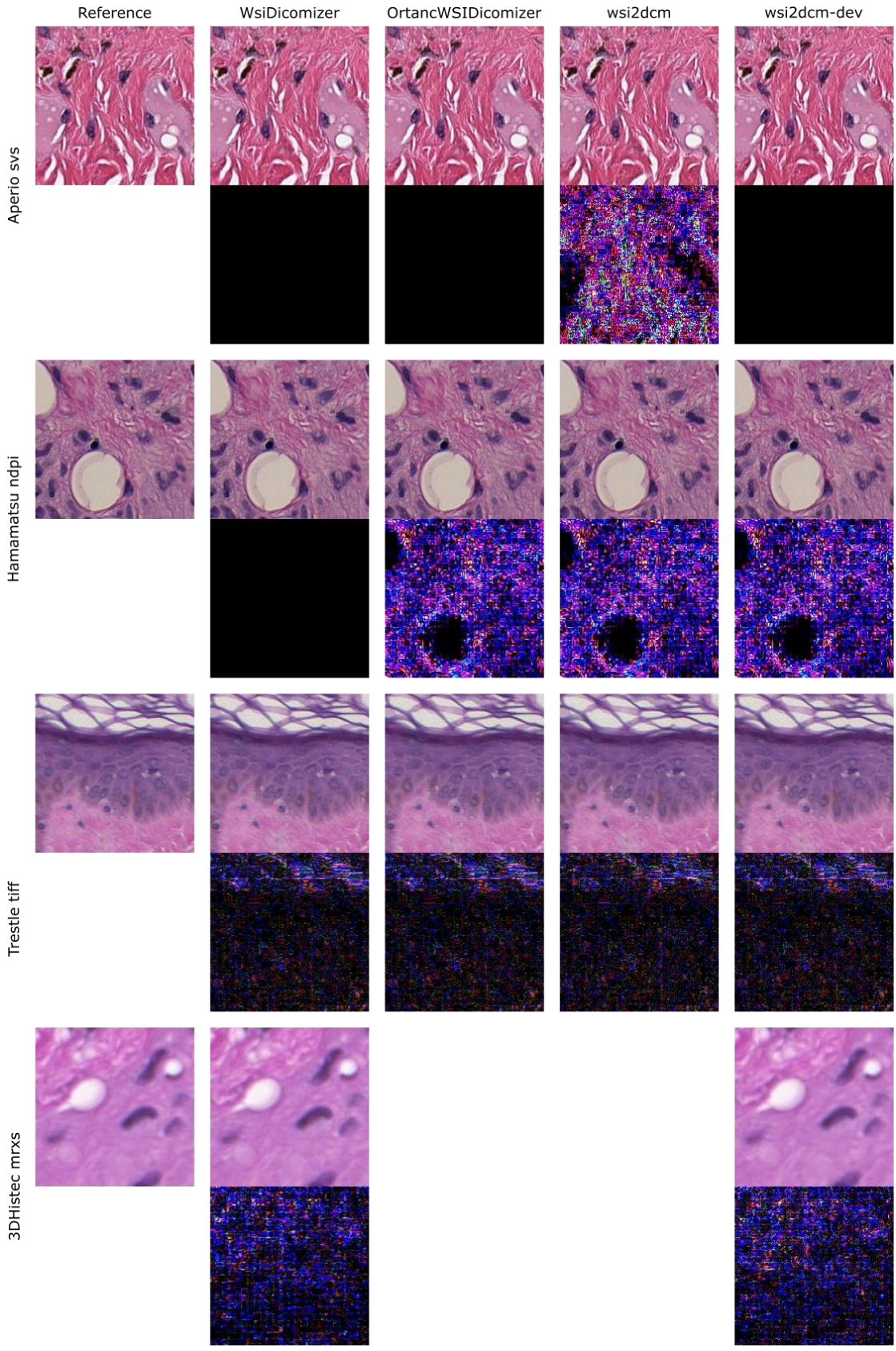

Figure 5. Visualization of the conversion loss. For each format the same region was read out from the source image (reference) and converted images (shown in the top), and the difference to the reference image calculated (shown in the bottom). The difference images have been contrast enhanced.

Conversion time

The mean time needed to convert a WSI file using the tested converters is presented in Figure 6. The conversion time is primarily dependent on how large and complicated the source image format is to read, and whether the conversion can be done lossless or not, but also how optimized the converter is.

All converters show the best performance for Aperio SVS-files. As discussed in the section *Conversion loss*, OrthancWsiDicomizer, wsi2dcm-dev, and WsiDicomizer can all lossless convert Aperio SVS-files, with wsi2dcm-dev being the fastest with approximately 0.7 s, followed by WsiDicomizer at 3.5 s and OrthancWsiDicomizer at 6.9 s. As only minimal processing of the tiles is needed before writing to the converted file, it is likely that how optimized the IO-procedures are the primary performance metric in these cases.

Apart from longer conversion time for wsi2dcm, the converters perform similarly for the Trestle format. Here all converters employ re-encoding, and it is likely that the performance is limited to how optimized the Jpeg-compressor is.

For Hamamatsu NDPI, only WsiDicomizer performs lossless conversion. Unlike the lossless conversion for e.g. Aperio SVS, this conversion is involves more image processing, as the striped image data must be re-formatted into rectangular tiles, resulting in a in comparison longer conversion time. Other converters perform re-encoding of the image, resulting in at best approximately double the conversion time.

The longer conversion time for the Mirax format is likely attributed to the large size of the image. Additionally both OrthancWSIDicomizer and wsi2dcm failed to convert the file due to memory exhaustion, like as they do not interpretant the imaged region correctly.

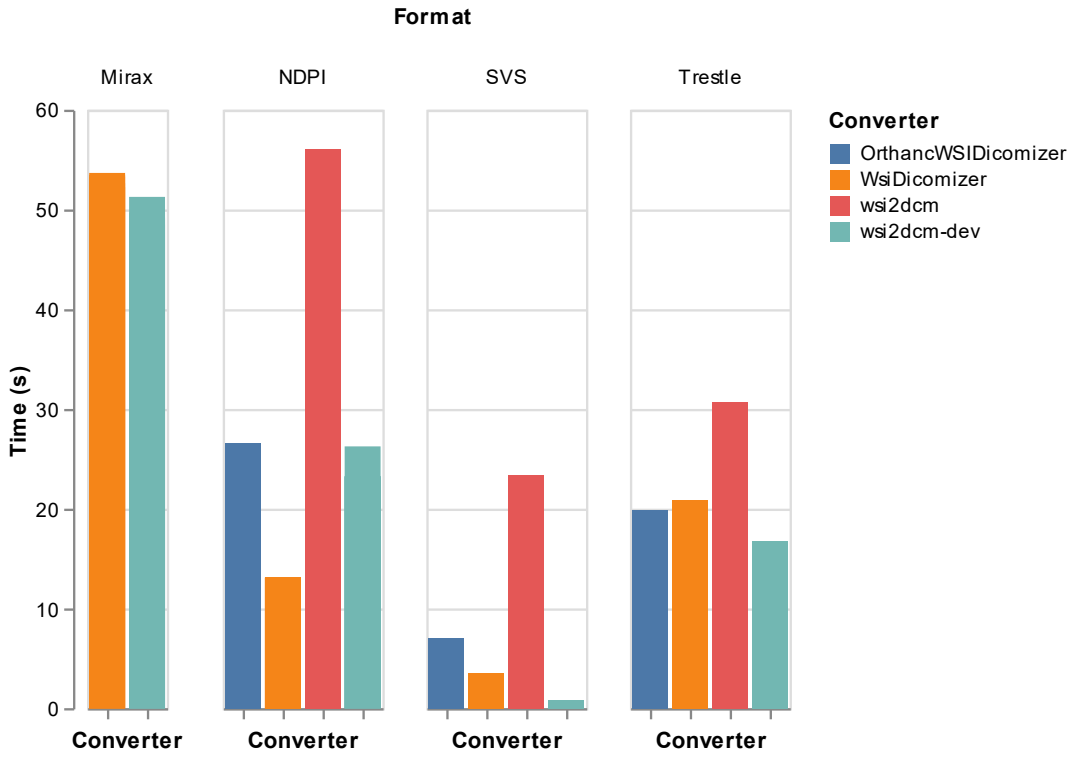

Figure 6. The mean time needed to convert a WSI file into DICOM WSI based on converter software used and grouped by input WSI file format.

Loading time

The time to load and read a single tile from DICOM WSI files converted using different converters and settings is presented in Figure 7. The fastest loading time is observed files converted using WsiDicomizer with either BOT or EOT. The type of tiling and presence of offset table in the converted files, as presented in Table 6, explains the observed variations in loading times.

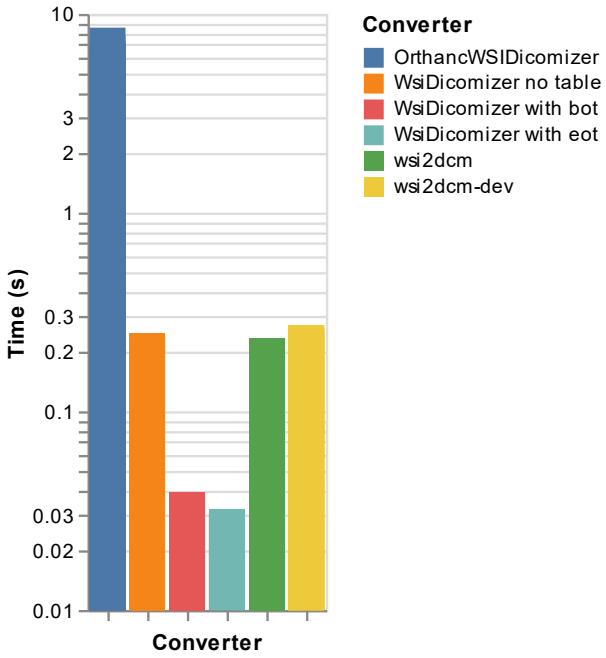

Figure 7. Loading time of files converted using tested converters and settings.

When using full tiling, a tile's order in the file can be directly calculated from the tile's x, y, and z-position and optical path. For sparse tiling, the file contains is a sequence of records describing the x, y, and z-position and optical path for each tile present in the file. To get the order of tiles in the image file this sequence must be parsed. Unfortunately, the structure of the record prevents efficient parsing, and the loading time is thus increased significantly.

Each tile in the file is encapsulated as an *item*, which starts with a tag followed by the item's length (including the tile). When there is no offset table present, the location and length of each tile in the file must be obtained by reading the position of the item, reading its length, and jumping to the end of the item. The reading is thus non-sequential. When an offset table is present, the position (and for EOT also length) is recorded in a separate table that can be read sequentially, resulting in faster performance. Although the size of a EOT is double that of a BOT (64 bit values compared to 32 bit values) the loading time is not significantly different.

Table 6. The observed tiling type and presence of table offset in converted files.

Converter	**Tiling**	**Offset table**
WsiDicomizer with BOT	Full	BOT
WsiDicomizer with EOT	Full	EOT
WsiDicomizer without table	Full	None
OrthancWSIDicomizer	Sparse	BOT
wsi2dcm	Full	None
wsi2dcm-dev	Full	None

Discussion

Anonymization

The Description of the action (DoA) states that tools for anonymization and pseudonymization of DICOM WSI files are to be developed. Such tool has not yet been developed as:

- It has been decided that most metadata are to be provided in dedicated metadata files and not stored in the DICOM image files. Thus, for BigPicture use, there is limited number of metadata to anonymize.
- When the tool is used in a BigPicture workflow, required DICOM metadata is provided from the (anonymized) BigPicture metadata.
- The tool allows sensitive metadata, and label and overview images (that can potentially include personal or sensitive information) present in the source image file to be excluded in the conversion.

Work is ongoing to provide improved image anonymization by implementing overlays to be applied to label and overview images, thus enabling sensitive label and overview images to be converted with sensitive information blanked out or replaced.[22]

If in the future anonymization or pseudonymization of already existing DICOM WSI files are needed, a solution using already existing open-source tools such as deid[23] could be implemented with relative ease.

Support for other formats through Bio-Formats and other proprietary formats

The DoA states that existing open-source libraries and vendor-provided SDKs should be used where possible to implement the DICOM converter. For example, the library Bio-Formats provides a collection of image readers that could be used to support even more image formats. Using Bio-Formats has however not been possible due to license incompatibilities. To allow the BigPicture software results springing from the work described in this deliverable to be used as freely as possible, the permissive Apache 2.0-license was chosen. Most of the interesting image readers in Bio-Formats are licensed under GPL,[24] which requires any software that uses the image reader to also be distributed as GPL. Distributing an Apache 2.0 project using GPL software is thus not possible.[25] Technically it is however possible to use either the BSD or GPL-versions of Bio-Formats to enable the converter to read additional formats, and the BSD version might be included in later

[22] https://github.com/imi-bigpicture/wsidicomizer/issues/34

[23] https://github.com/pydicom/deid

[24] https://www.openmicroscopy.org/licensing/

[25] https://www.apache.org/licenses/GPL-compatibility.html

releases.

So-far no vendor-provided SDKs has been used to implement support for image formats, primarily as no interest for such implementation has arisen within the BigPicture community. However, the design of the converter enables support for additional formats using image data read through external libraries to be implemented with relative ease.

As an alternative to using SDKs to directly read image data from proprietary formats, one can also use vendor-provided tools to convert the image to a supported format, for example TIFF or DICOM. The presented tool can then be used to convert or re-write the image into the BigPicture DICOM WSI format. This has been done for Philips iSyntax-files and 3DHistech MRXS-files.

Conclusion

As the chosen WSI format for the BigPicture project is DICOM and that most image submitting partners in BigPicture have images in other formats, a tool for converting WSIs to DICOM is needed. Existing converters provide good basic support for most common WSI formats but does not implement all the settings and features that BigPicture has decided to use in the DICOM format, prompting the development of a new converter fitting the BigPicture needs. The converter presented in this deliverable:

- Can produce DICOM WSI files according to the decided BigPicture standard.
- Offer lossless and fast conversion whenever the format supports it and ‘as good as possible’ conversion for other formats.
- Is extendable to other not-yet supported formats.
- Integrates with BigPicture metadata, both by inserting BigPicture metadata into the converted files and by providing image metadata to the BigPicture metadata.
- Is free to use under a permissive license and open to open-source community contributions.

The converter has already been used by several partners to convert WSI into DICOM WSI with success, for example to create pilot datasets.