OSSI proposal: Paintera

Caleb Hulbert, John Bogovic and Stephan Saalfeld

Motivation

The Paintera tool\(^1\) enables dense annotation of large 3D volumes. It uses the ImgLib2-based BigDataViewer rendering engine\(^2\) to virtually slice and zoom into arbitrarily large composites of 3D volumes. The N5 API\(^3\) is used to store and load large multi-scale raw volumes, labels, and meta-data in HDF5 files, N5 file stores, or in the cloud. Paintera is implemented in Java\(^4\) and Kotlin\(^5\) and uses JavaFX\(^6\) for its user interface which provides hardware accelerated 3D rendering on all major platforms. At this time, Paintera features 2D and 3D label brushes and interactive shape interpolation in arbitrarily re-sliced orientations, tools for manual superpixel agglomeration, and on-the-fly mesh generation and interactive display with adaptive level of detail (LoD) and smoothing.

Paintera evolved from a series of ad-hoc solutions to urgent real world issues and therefore has an organic history. Initially, the Saalfeld lab, in collaboration with then visitor Jan Funke and the Bock lab, needed a tool to rapidly annotate dense ground truth for neuron and synapse segmentation in non-isotropic serial section transmission electron microscopy (ssTEM) if the *Drosophila* brain.\(^{7}\) We had first used FlyEM’s Raveler tool\(^7\) to annotate a sample with mostly orthogonally sectioned neurons. Raveler, however, does not include 3D visualization, reslicing, and other means to assess 3D continuity of 2D annotations and therefore the labels included a large number of annotation errors. We then contracted the company ariadne.ai\(^8\) whose undisclosed workflow improved upon Raveler generated labels but still contained too many obvious errors to be practically useful. We reasoned from ongoing communication with ariadne.ai that, at the time, they were unable to handle larger volumes efficiently and had no effective tool to validate 3D continuity of labels. We therefore used BigDataViewer as a platform to developed the annotation tool BigCAT\(^9\) which uses the HDF5 storage format for raw volumes, labels, and annotations and can lazily load only the parts of the raw volume that are in the field of view (FoV). We developed manual agglomeration tools to interactively merge and split segmentation

---

\(^1\)Paintera: https://github.com/saalfeldlab/paintera
\(^3\)N5: https://github.com/saalfeldlab/n5
\(^4\)OpenJDK: https://openjdk.java.net/
\(^5\)Kotlin: https://kotlinlang.org/
\(^6\)JavaFX: https://openjfx.io/
\(^7\)Raveler: https://wiki.janelia.org/wiki/display/RAV/Raveler
\(^8\)ariadne.ai case study: https://ariadne.ai/case/segmentation/cells/MICCAIChallenge/
\(^9\)BigCAT: https://github.com/saalfeldlab/bigcat
Figure 1: Paintera running on the Ubuntu Linux desktop was used to proofread automatic neuron segmentations and pre- and post-synaptic contact predictions in non-isotropic ssSTEM of the *Drosophila* brain to improve the CREMI challenge.

fragments, and 2D and 3D label brush and fill tools. BigCAT was used by a small number of annotators in the lab and associated groups to finalize the annotations at a significantly smaller error rate than before and we eventually used it to host the CREMI challenge.

The CREMI challenge data enabled several groups to dramatically improve the performance of neuron segmentation pipelines\(^{10}\) and the detection of synaptic contacts\(^{12}\) which, in turn, revealed that more and even better curated training labels are required to train and validate machine learning based reconstruction methods. In parallel, other groups, e.g. the COSEM team\(^{13}\) started using BigCAT to fine-tune their training labels for cellular organelles in isotropic FIB-SEM. BigCAT lacked three main features to support these efforts more efficiently: (1) lazy-loading and -writing support for multi-scale labels, (2) interactive 3D rendering of selected labels, (3) 2D label brushes and fill tools that work in arbitrary orientation instead of just the orthogonal axes, (4) interactive interpolation between arbitrarily oriented shape profiles labeled at a distance. For its platform independent support of hardware accelerated 3D rendering, we switched to the JavaFX platform and developed the Paintera tool which now supports all of those features. We also replaced the HDF5 backend by the N5 API which makes it easier to use Paintera on both file-based back-ends (including HDF5) and cloud storage. Paintera has been used successfully to densely proofread volumes that were more than 100× larger than the CREMI challenge volumes.

\(^{10}\)CREMI: https://cremi.org/

\(^{11}\)Plaza and Funke 2018; Cerrone, Zeilmann, and Hamprecht 2019; Luther and Seung 2019; Lee et al. 2019; Funke et al. 2019; Li et al. 2019; Bailoni et al. 2019; Linsley, Kim, Berson, and Serre 2020; Lee, Lu, Luther, and Seung 2021.

\(^{12}\)Heinrich et al. 2018; Buhmann et al. 2021.

\(^{13}\)Heinrich et al. 2020.
Figure 2: Paintera running on the Windows desktop was used to generate training labels for Starfinity cell body segmentation in large light sheet microscopy volumes from mouse cortex and to visualize and proofread the results. (see Fig. 1) and to create training labels for Starfinity cell segmentation in near isotropic light sheet microscopy volumes\(^{14}\) (see Fig. 2).

**Goals and milestones**

While Paintera has been demonstrated to be useful as is, it carries some historical legacy and lacks useful features to make it an even better tool and to enable its inclusion as a module for future active machine learning tools for ongoing projects for CellMap and the 4D Cellular Physiology research area.

**Database for indices:** Paintera maintains indices to accelerate caching, mesh generation, updates, and search queries over labels and fragment agglomeration. These indices are currently stored as N5 datasets but the N5 format is not ideal for this purpose because it creates a significant number of files and/or has high latency for search operations when stored in the cloud. We have hesitated to add more indices that would be useful to support targeted proofreading workflows because of these technical difficulties. The appropriate store for these indices is an SQL database such as SQLite, PostgreSQL, or MySQL\(^{15}\). Paintera should connect to and use an SQL database to maintain its indices and other relational data. Ideally, a Paintera project without a dedicated database server would store a compressed dump of the database content in its project N5 file and create and fill a local SQLite instance when

---

\(^{14}\)Wang et al. 2021.  
\(^{15}\)SQL databases: [https://www.sqlite.org/](https://www.sqlite.org/) [https://www.postgresql.org/](https://www.postgresql.org/) [https://www.mysql.com/]
the project is loaded. Projects that connect to a dedicated database server will store
the necessary account information.

**Improve mesh generation:** Paintera supports interactive mesh generation with view depen-
dent LoD. Meshes are generated for label blocks, not for objects, and smoothed
to improve visual appearance. Label blocks currently overlap significantly to guar-
antee visual continuity. This overlap is not required if mesh smoothing operates on
sufficient padding to guarantee that corresponding vertices undergo the same oper-
ation. Furthermore, the current implementation does not re-use vertices for adjacent
polygons but naively duplicates them. This leads to a more than 6× inflated memory
demand on the GPU. Lastly, the parameters for mesh smoothing do not currently
consider the LoD at which the mesh is being rendered. This leads to over-smoothing
at lower LoD and inconsistent appearance of the meshes at detail level transitions.
We have already implemented the improved smoothing method including LoD depen-
dency and adjusted the code to re-use vertices instead of duplicating them, however,
this code has not been integrated into Paintera yet. Integration is not entirely trivial
because details of the previous method are deeply mixed with other components of
the tool, so some careful testing and refactoring will be required.

**Modularize:** Paintera’s current tool set and UI are rather monolithic and almost all com-
ponents interact with each other. This simple design makes it easy to serialize and
deserialize the state of a project but it is difficult to use individual components of
Paintera in other projects. We plan to better separate individual components and
their interfaces, to separate tools and UI control elements, and to make the UI more
flexible. We will convert control elements that are currently statically associated with
properties of sources, meshes, and tools into re-usable components that can be con-
ected to groups of elements. This will make it simpler to e.g. share mesh property
controls for several mesh sources and to de-clutter the UI. We will also convert the
currently static UI arrangement to a flexible tile tree like in the CATMAID window
manager.16

**Tools:** Paintera is currently a single stateful annotation tool that enables label painting,
manual fragment agglomeration, and label profile interpolation simultaneously. For
label profile interpolation, the capabilities of this global tool need to be managed
because both navigation and interaction with source controls must be constrained
while profiles are being selected. Anticipating the development of more tools, we
will introduce explicit control modes for stateful tools instead of depending on tools
to manage the entire UI. This will not only make the operation of the current set of
tools and states simpler but it will enable us to quickly add simple new tools that have
long been missing. We will start by adding contrast based magic wands, interactive
thresholding, seeded level sets, and label shape smoothing. We will also add an al-
ternative label shape interpolator that uses polygonal re-sampling of the label shapes
instead of boundary distances. This shape interpolation mode will make it simpler to
annotate fine structures like microtubules and thin membranes.

**Deployment and integration:** Currently, Paintera is released through conda, including a
JGO based launcher. We would like to switch to a more OS native release of both
Paintera and future integration with other tools and have submitted a parallel pro-

---

Proposal to improve the deployment of JVM based tools at Janelia and in the open source community. As Paintera is becoming more modular, it will be even more relevant to release its components and build and release various tools that re-use these components.

Requirements and synergies

Currently, the Saalfeld lab is developing Paintera as a tool to support ongoing projects in the lab and associated projects such as COSEM and CellMap. We cover development, user support, and deployment with one full time software developer and occasional associated work by other lab members. In particular the developments on high impact infrastructure components is secondary to urgent daily needs in the lab (this has not been the case for the last year but will become an issue as we transition back into normal research operations). The utility and applicability of Paintera would therefore strongly benefit from additional support by which the tool can become a very flexible component for semi-automatic proofreading methods on large volumes that we plan to develop in collaboration with the Funke lab. This will be particularly relevant for the CellMap project associated with 4DCP, and improved neuron and synapse reconstruction and classification for MCN.

We would appreciate support by developers with strong experience with the Java programming language and the JVM in general. Knowledge of the N5 API, ImgLib2, BigDataViewer, and the Kotlin programming language can be helpful but can be acquired along the way.

References


