ImageJ2

Development Next Steps

Components of ImageJ2

- Major components of ImageJ2
  1) **Data model** – ij.process
  2) **Display** – ij.gui
  3) **Input/output** – ij.io
  4) **Regions of interest** – various
  5) **Scripting & plugins** – ij.macro
- All of these areas have significant limitations and will benefit from enhancements and refactoring

Components of ImageJ2

- Define/refine API for each component
  1) **Data model** – ij.process→imagej.model
  2) **Display** – ij.gui→imagej.display
  3) **Input/output** – ij.io→imagej.io
  4) **Regions of interest** – various→imagej.roi
  5) **Scripting & plugins** – ij.macro→imagej.scripting

Defining the API

- Work together to draft the new API
- Once a week, spend a few hours in a room together
  - Define service interfaces (OSGi-style)
- Then, each tackle different packages
- Keep each other apprised of progress (dev. Cycles)
- Iterate the design work as needed

Summary of Goals

Data Model

- [SLIM] N-dimensional data model (e.g., SLIM)
- [Fiji] Additional data types (imglib; avoid case logic)
- [TrakEM, Hessman] Coordinate systems (registration)
- [Landini] Color space support (e.g., HSB)
- [µManager] Support for gamma (imglib?)

Display

- [VisBio] Tiled viewer (large planes)
- [VisBio] Improve VirtualStack caching (from BF)
- [VisBio] Improve 3D Viewer (arbitrary slicing)
- [TrakEM] Better ImageWindow (multiple data objects)
- [μManager] Fix brightness/contrast
- [μManager] Better histogram display (JFreeChart?)
- [μManager] Integrate Image5D features into hyperstacks

I/O

- [Bio-Formats] Unified I/O service for file types
- [VisBio] Tiled VirtualStacks (large planes)

ROIs

- [TrakEM] Vector-based ROIs (not always bitmasks)
- [TrakEM] Better support for multiple ROIs
- [TrakEM] More flexible usage of ROIs/masks/thresholds
- [Doube] N-dimensional ROIs
- [Tinevez] Better GUI controls for ROIs (JHotDraw)
- [Tinevez] Separation of ROI data vs. display (OME?)

Architecture

- [Fiji] Documentation mechanism (annotations)
- [Fiji] Modular command framework (to record scripts)
- [Fiji] Remove AWT dependencies (better headless)
- [μManager] Clearer API for plugins (service interfaces!)
- [μManager] Hooks to extend the GUI (e.g., hyperstacks)
- [μManager] Use a framework enabling MDI model
- [Misc] Create an all-Swing GUI?
- [Compatibility] Existing code delegates to new code

Interoperability

- [FARSIGHT] Call ITK & FARSIGHT from Java (imglib?)
- [CellProfiler] Combine ImageJ and CellProfiler workflows
- [OMERO] Import/export data to/from OMERO database

Next Steps

Next Steps: Data

- Refine imglib library
  - Gamma
  - Coordinate systems (e.g., affine transforms)
  - Tiles and caching
- Refactor ImageJ to use imglib Image instead of ImagePlus
- Will we need any layers on top of imglib Image?

Next Steps: Display

- Define interface for improved image viewer
  - N-dimensional
  - Support for tiles
  - Support multiple data objects (imglib images, ROIs)
  - Choose Image5D features to integrate
  - Design with extensibility in mind
- Begin work implementing the interface
- Define interface for histograms
- Implement better histogram support with JFreeChart

Next Steps: I/O

- Split Bio-Formats into core and readers modules
  - BF-core module will be BSD
  - BF-readers module will stay GPL
  - BF-core can then be bundled with ImageJ
  - BF-core will read/write OME-XML and OME-TIFF
  - BF-core will support all ImageJ “out of the box” formats
  - BF-readers will cover microscopy-specific formats
- Develop a service-based mechanism for modular file format support
- Migrate existing ImageJ formats into BF-core

Next Steps: ROIs

- Define interfaces for ROI hierarchy
  - Use existing OME class hierarchy if possible (ome-xml.jar)
  - To use ome-xml.jar, license must be compatible
  - Separate ROI data (algorithms) from ROI display (overlays)
- Use JHotDraw for ROI display and manipulation
  - But look for ways to preserve existing ImageJ ROI UI

Next Steps: architecture

- Integrate annotation-based plugins mechanism
  - Upgrade existing core plugins, as examples
- Split ImageJ into modular components
  - Use OSGi services as demonstrated by Rick & Grant
  - Clearly delineate “external API” via appropriate interfaces
  - Establish repository structure to match, using Maven
  - Document development best practices on imagejdev.org
- Define interface for commands
  - Needed for recording scripts
  - Separates analysis workflow from UI

Next Steps: architecture

- Identify specific AWT dependencies
  - E.g., GenericDialog
  - Rearchitect to eliminate them
- Rework legacy code (ij.*) to call into new interfaces
- Application framework
  - Internationalization
  - Swing
  - MDI