Theba Documentation

Jens Bache-Wiig Per Christian Henden

October 22, 2007

Contents

Chapter 1

Introduction

The primary goal for Theba has been to develop and implement a practical tool to aid the analysis of three-dimensional images of fibrous materials.

In [2] a method individual segmentation of fiber images was suggested. Our goal has been to improve and continue this work.

Different fibrous materials tend to have widely different characteristics, and most algorithms fail to cover all cases. To make these tools practical for future use, emphasis has been put on expandability, re-usability and configurable solutions.

Chapter 2

User guide

This section aims to give new users a short introduction to the Theba application.

2.1 Introduction

The Theba application is an interactive tool as well as a programming framework to aid material scientists in the segmentation and analysis of three-dimensional volumes. It is primarily made for wood fiber segmentation, but efforts have been made to keep the base architecture as open, modular and reusable as possible to enable expansion and experimentation in other areas as well.

2.2 System requirements

1. Sun or Apple Java 1.5.0 or later (also called Java 5.0)
2. 256 MB free main memory (512MB recommended)

2.3 Licensing

The Theba application is in the public domain.

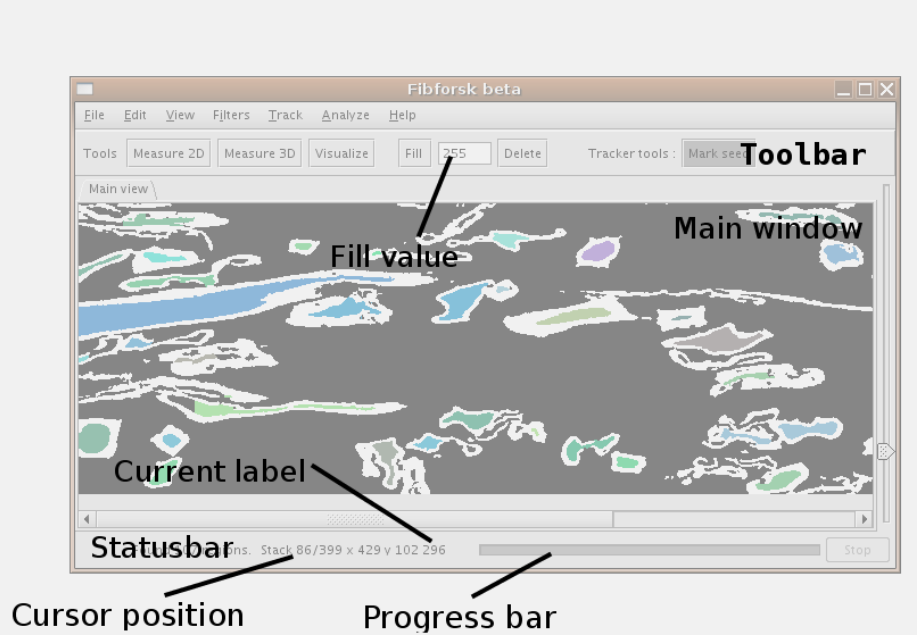

Figure 2.1: Overview of the Theba interface

2.4 Installation

Installation is simply a matter of copying the Theba zip-file to your computer and extracting it.

2.5 User interface

Figure 2.1 shows the main components of the GUI interface. The *Toolbar* is contains a series of toggle-buttons, indicating the currently selected tools. The currently available tools are :

- Measure 2D : Applies and prints results from all available 2D measurement descriptors to a selected -connected 2D region
- Measure 3D : Applies and prints results from all available 3D measurement descriptors to a selected labeled region
- Visualize : Opens a visualization of the selected labeled region in a separate visualization window

- Fill : Applies 26-connected three-dimensional flood-fill to the selected voxel using the indicated *Fill value*.
- Delete : Applies 26-connected three-dimensional flood-fill with the label 0, effectively deleting the region.

Note that other tools may be added or removed depending on the currently selected Tracker-plugin.

The *Main Window* contains a 2D window into the volume. The user may scroll through the volume by using the *depth slider* on the right hand side of the main window.

The *Status bar* indicates the cursor's current x,y,z position, as well as the value/label of the voxel directly below the cursor cross-hair. The *Progress bar* indicates the status of the current process. When a process is running, the *Stop* button may be pressed to terminate it. Note that this is not equivalent to canceling the action, and pressing this button can leave the program in an unstable state.

2.6 Performing individual fiber segmentation

The work flow for segmenting a binary volume in Theba consists of a series of steps required to successfully segment the volume.

The volume is first binarized. Since this procedure is highly data dependent, we have not included this functionality in the Theba application. A method for binarization of 3D fiber volumes are explained in [1], chapter 4.2.

This work flow describes the main steps involved in correctly segmenting a fibrous volume using the Theba tool and the algorithm described in [1], chapter 5.2.

2.6.1 Starting up

Theba is loaded by executing the include jar-file (theba.jar).

Windows: If you got 768MB or more memory: just double-click the theba jar-file. Otherwise, execute it using 'java -Xmx256m -jar theba.jar' from the command line.

Unix: Execute it using 'java -Xmx512m -jar theba.jar' from the command line. Make sure 'java' is Sun or Apple Java. Change the -Xmx setting if you have less than 512MB RAM available to Java, to for example -Xmx384m.

It is important that enough free memory is available to Theba. By default all slices will be loaded into the primary memory. If not enough memory is available, extensive disk-swapping will make most operations inefficient. However, it is

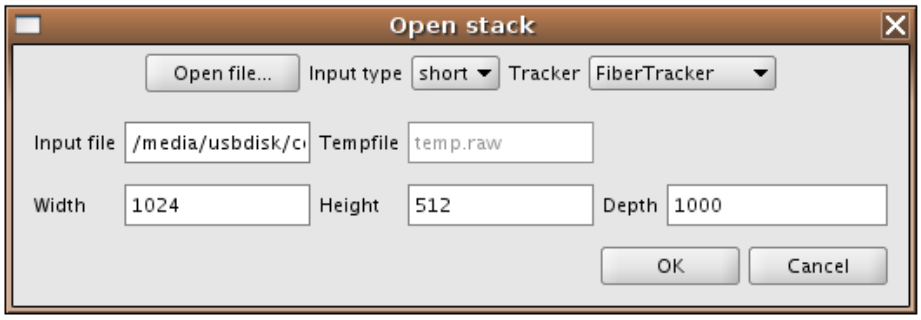

Figure 2.2: The opening dialog in Theba

possible to enable a special Slice-based cache that will allow the fiber-tracking to execute efficiently even though the host computer cannot hold more than a few slices in memory at once. To do this, open the preferences panel from the *edit*-menu and set the *use_cached_access* property to 1. The user may safely leave the temp-file settings with the default value, unless there is little storage space left on the host hard drive, in which case he must reduce the numbers.

2.6.2 Workflow

1. The open dialog shown in figure 2.2 will automatically open upon start, and can be also be reopened with *File→Open*. The user must indicate the correct data type, width, height and depth of the binarized volume. Note that all previously saved volumes will always be in short-format. He must also ensure that the *FiberTracker* is selected from the tracker selection menu.
2. The user may choose to do the segmentation manually, however it is recommended that *auto segmentation* is selected from the menu first. This uses the method suggested in [1], chapter 5.2, to pick up the best candidates in evenly spaced intervals along three spatial directions separately. Depending on the quality of the binarized volume this will often detect most of the candidates.
3. After the auto tracking is complete, the user inspects the volume using the depth slider on the right hand side, and seeds undetected fibers if necessary using the *Mark seed*-tool on the toolbar. Erroneously detected lumens can be removed from the 3D volume by using the delete-tool available on the tool bar, and regions can be joined by using the Fill-tool on the toolbar. Right-clicking a region automatically sets the fill-value to the label of the clicked region. When enough seeds have been selected, the *Track→Track lumens*-option initiates tracking in both directions for each of the seeded

lumens. The seed-list can be cleared by selecting *Track*→*Clear seeds* as well.

4. Flip the volume along the two other principal axis and repeat the previous step until no more lumens can be found.
5. After all traceable lumens have been isolated, the final segmentation-step can be applied by selecting *Track*→*Track walls 3D* from the menu.
6. Save the final segmented volume by selecting *File*→*Save*.

Chapter 3

Architecture

This section aims to give a brief overview on the design goals and implementation details of the Theba application.

3.1 Requirements

The purpose of this application is to be a practical tool for fiber segmentation and analysis. The tool should be able to analyze, segment and present usable statistics on binary data. This section will present the requirements we used as a basis for our design.

3.1.1 Primary requirements

1. Binarization of fibrous data
2. Automatic detection of individual paper fibers
3. Segmentation of individual fibers
4. Analysis of this data

3.1.2 Secondary requirements

1. The software should be able to load raw binary data in short- and byte-format.
2. Segmented volumes must be loadable by the ImageJ-platform.
3. The tracker-components must be interchangeable and separated from the user-interface

4. The paper tracker must support data analysis of segmented fibers including estimates for
   (a) Fiber length
   (b) Fiber size in voxels
   (c) Fiber surface area
   (d) Fiber orientation
5. The software must be able to run on both Linux- and Windows platforms, having a minimum of 512 MB primary memory (256MB available).
6. The software should have a modular architecture, allowing software to be reused and developed to support different segmentation algorithms.

3.2 Development platform

The Theba application was developed in Java to ensure portability. Sun or Apple Java 1.5 or later is required.

3.3 Base architecture

Theba is a stand-alone application with a graphical user interface. While it was initially intended to be an extension to the public domain ImageJ image analysis environment, this did not give us enough flexibility to experiment with user interaction in the process of tracking fibers. Another important reason for not using ImageJ as the host environment was the need to use a more flexible storage and data management, than what was possible with ImageJ. Loading datasets using ImageJ requires the entire volume to be resident in main memory at once. This places strict requirements on primary memory as well as restrictions on the maximum size of the volumes to be analyzed.

Theba is linked to libraries provided by ImageJ [4] as well as Jama [5]. These libraries are in the public domain. It also links to a Watershed plugin by Christopher Mei [3]. This library is under the LGPL license, making it free to use, but requires that changes to it are made official if they are to be distributed. No changes should be necessary.

3.4 Package and class-overview

Theba is the name of the application we have developed, to perform fiber segmentation as well as data analysis.

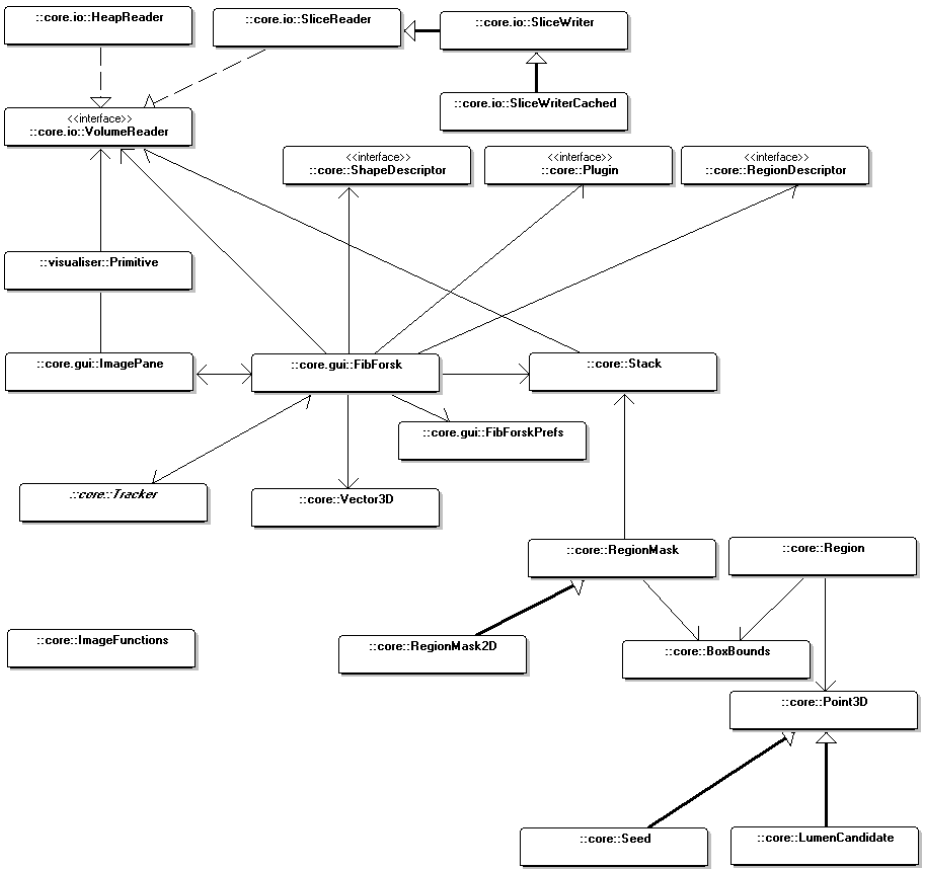

Figure 3.1: UML class diagram for the Theba application

The Theba code base is divided between the main packages core, descriptors and trackers. The core-package consists of most of the core functionality of Theba, including the user-interface (core.gui), IO (core.io) architecture and reusable image processing libraries (core.Imagefunctions, Watershed etc). A simplified UML class-diagram is presented in figure 3.1.

3.5 Storage

This section presents a description on the file-formats and methods for file access used in the Theba application.

3.5.1 File format and output

Save-data is exported to short-valued little-endian raw-format, appending an empty byte when loading and saving single byte raw volumes.

We have reserved the range 0–255 to indicate gray-level information in the data, and such values will be indicated by gray-levels within the Theba application. These do not indicate segmented regions, and are preserved when saving a data volume to file. Their primary use is for loading original gray-scale data or distance maps as input to the segmentation algorithms. This implies that loading byte-based images and saving them, will always result in an equivalent short-valued image. By default this will appear as a dark gray-scale image when exported to other image-applications, since we have effectively only padded an extra byte to each byte in the input.

Any values from 256 and up to 8192 may be used for labeling, giving a maximum of 7196 separate regions/fibers by default. The maximum fiber count is currently 4096, but this number can be increased by adjusting the *maxFiberValue* property in the properties dialog at the expense of slightly higher memory requirements when performing measurements on segmented regions.

This leaves the three higher-order bits reserved to serve as internal temporary flags by any tracking routines.

3.5.2 File input

The Theba application can import data in RAW byte- or short-format. Each voxel is packed in a row-by-row, slice-by-slice order. When short-data is read, the lower-order byte comes first (little endian).

3.5.3 File management and cache

For segmentation purposes we require fast read- and write-access to data on a pixel-by-pixel basis. This makes most compression-schemes unfeasible. In addition, all segmented regions require a separate label. Since the amount of fibers in a typical paper volume easily exceeds 256, the maximum value available within a single byte, we use two bytes per voxel for all data-storage internally.

All data access is handled by the *VolumeReader* interface. We have two different implementations of this interface, namely *SliceReader* and *HeapReader*. Both give the same basic functionality, but can be interchanged depending on the type of data access required and the amount of available memory on the host computer.

The SliceReader caches separate image-slices in a FIFO buffer, not requiring the entire data structure to remain in memory at once, effectively moving a slicing

window of data across the volume. Since our fiber-tracking routine operates in a linear fashion through the volume this form of cache can be efficient.

The HeapReader implementation, works by loading all slices into heap memory, or on disk using any built-in swapping algorithms provided by the host operating system. This latter method is preferred if the host platform has enough primary memory to support it, and is used by default. To change this behavior, set the preference *use_cached_access* to *1*, under the *edit→preferences* menu.

The Stack class is another abstraction layer to the data, giving the user access to individual voxels, size, orientation and other information regarding the data.

3.5.4 Caching

Efficient caching is essential to avoid performance loss, as access to secondary storage is very slow. Some caching is already provided by the hardware, as well as the operating system. However, these caching algorithms do not employ any knowledge about the data we are accessing and we have therefore developed a simple, yet effective cache-scheme optimized for our particular tracking-method.

The type of cache that we have implemented is based on the fact that our primary tracking method generally requires a slice-by-slice access. Access to individual imageSlices, has therefore been optimized. We have developed a set of classes, SliceReader and SliceWriter enabling us to randomly access 2D-slices from a 3D volume on disk using a moderately large LIFO-based write-back cache.

This implies that changes to a cached slice, will *not* be reflected on disk unless the particular slice is replaced in cache or the writer is explicitly flushed. All algorithms employing the SliceWriter class should therefore be written to ensure that such caches are flushed explicitly after completion, by calling the *<SliceWriter>.flush()*-method to ensure that results are properly stored to disk.

The actual replacement policy can easily be modified by overloading the method updateCache() in the *SliceWriter*-class. The amount of slices to store in cache is configurable by changing value of the *defaultSliceCount*-property from the preference-panel on the edit-menu.

3.6 Trackers

The tracker is the class responsible for the actual segmentation. Any segmentation routine depending on domain knowledge will have to be altered when applied to a different type of data, even though a lot of the functionality required is similar. When trying out different approaches on the same data it

is also practical to be able to compare the performance of different tracker-methods. Thus we have made the tracker a pluggable interface in the Theba framework.

The tracker has access to the volume-data, the user libraries provided by Theba, as well as those provided by ImageJ.

For a complete description of the Tracker-interface, consult the source code of the file core/Tracker.java.

3.6.1 Implementing and installing a new tracker

A new tracker-class can be created by implementing the Tracker interface located in the core-package. The resulting class-file must then be placed in the *trackers* package. Upon restart of the Theba application, the new tracker will automatically be detected and available for selection in the Open-dialog.

3.6.2 Storage of volume-data output during tracking

When tracking fibers, it is often useful to create and compare several data volumes as output of any given algorithm. This makes it possible to debug graphically what is happening at several different stages, as well as storing temporary structures that can be re-used for other purposes. We have developed a programming-interface to present the contents of a particular SliceWriter to the main-window in Theba, resulting in a separate tab in the main window. This makes it easy to switch back and forth between the separate output volumes for comparison. All these windows must have the same dimensionality as the main volume.

Example usage This particular example shows how to connect the output of a particular SliceWriter to the main application window, so that the user can select it while browsing through the volume and compare it to the other visible data-channels.

```
SliceWriter lumenlog = new SliceWriter("lumens.raw", width,
height);
Theba.getInstance().addChannel(lumenlog, "Lumens");
```

3.7 Descriptors

Analyzing a segmented volume is done by applying a set of measurements, such as size, orientation, curvature etc. These do not change the volume in question but usually return numerical data on different properties. We have introduced

a particular interface for exactly this purpose, making it possible to easily add new measurements to the already implemented modules by implementing only the relevant descriptor-module.

Descriptors designed to work on three-dimensional data can be recognized by exploiting the fact that they implement the method does3D(). If a descriptor returns a single double-typed value, they implement the isNumeric() method. The latter makes the descriptor available to any function assuming a single-valued input, such as the automatic seeding algorithm presented in [1], chapter 5.2.

An *ArrayList* list containing all currently available descriptors can be obtained by calling the method *Theba.getDescriptors()*.

```
package core;
public interface RegionDescriptor {
    public Object measure(RegionMask mask); //Generates the
        descriptor result
    public String getName();  //Returns the name of this descriptor
    public String getAbout(); //Returns a short description of this
         descriptor
    public boolean does3D(); //Returns true if this descriptor is
        applicable on 3D data
    public boolean isNumeric(); /*Returns true if the measure
        returns a
        Double-object as its only result */
}
```

3.7.1 Implementing and installing a new 3D region-descriptor

Region descriptors can be used to generate statistics on segmented 3D data. Examples of descriptors are included in the *descriptors* package.

A new descriptor-class can be created by implementing the *RegionDescriptor* interface located in the core-package. To be available for 3D region measurements the method does3D() must be overloaded to return *true*. The resulting class-file must then be placed in the *descriptors* package. Upon restart of the Theba application, the new tracker will automatically be detected and made available for selection in the *Analyze→Region measurements* menu.

3.7.2 Using descriptors

When the descriptor is selected from the *Region measurements menu*, the data will be analyzed by creating a set of box-bounded *RegionMask* objects that are passed to the descriptor's measure()-method. The result for each labeled region will be listed in the console window of the Theba application.

3.8 Image functions

The *ImageFunctions* class contains java implementations of many of the algorithms presented in [1], chapter 3. Some of these methods are:

- 2D- and 3D flood-fill and masked flood-fill
- 2D- and 3D dilation, erosion and masked (geodesic) dilation
- 2D- and 3D distance maps
- 3D geodesic distance maps
- linear interpolation methods

3.9 Tracking Fibers

The FiberTracker class is an implementation of the Tracker interface, and represents an implementation of the ideas presented in [1], chapter 5. Only the components that directly affect paper fiber segmentation have been included. Complete source code will be made available online or by direct request to the authors.

Bibliography

[1] Per Christian Henden and Jens Bache-Wiig. Individual fiber segmentation of three-dimensional microtomograms of paper and fiber-reinforced composite materials. Master's thesis, Norwegian University of Science and Technology, 2005.

[2] Runar Holen and Marit Hagen. Segmentation on absorption mode x-ray micro tomographic images of paper. Master's thesis, Norwegian University of Science and Technology, 2004.

[3] Christopher Mei. Watershed plugin for imagej, 2003. `http://rsb.info.nih.gov/ij/plugins/watershed.html`.

[4] W.S. Rasband. Imagej, 1997-2005. `http://rsb.info.nih.gov/ij/`.

[5] The Jama team. Jama: A java matrix package, 1999-2005. `http://math.nist.gov/javanumerics/jama/`.