JLIFE: THE JEFFERSON LAB INTERACTIVE FRONT END FOR THE OPTICAL PROPAGATION CODE*

Anne M. Watson# & Michelle D. Shinn
Jefferson Lab, 12000 Jefferson Ave, Newport News, VA, 23606 U.S.A

Abstract

We present details on a graphical interface for the open source software program Optical Propagation Code, or OPC [1]. This interface, written in Java, allows a user with no knowledge of OPC to create an optical system, with lenses, mirrors, apertures, etc. and the appropriate drifts between them. The Java code creates the appropriate Perl script that serves as the input for OPC. The mode profile is then output at each optical element. The display can be either an intensity profile along the x axis, or as an isometric 3D plot which can be tilted and rotated. These profiles can be saved. Examples of the input and output will be presented.

INTRODUCTION

Since its creation in 2006 to model wave propagation in FEL oscillators, OPC has been used to simulate a wide variety of laser systems. Like most scientific programs, OPC is written such a way that a user interacts with the program using a script they must write themselves. Additionally, the output must then be plotted using a different scripting syntax. For those inexperienced with such coding work, it can be a daunting task, so they choose not to use the program at all. It was our desire to construct a user-friendly piece of software to handle the back-end portion of the optical calculation and present solely the simple forms of input and output. JLIFE, or the Jefferson Lab Interactive Front-End, enables a user to select and configure various optical elements within a web-like form interface. This data is then translated behind the scenes into the appropriate code for use with OPC, as shown in Figure 1. The resultant graphs showing the beam profile at each element are displayed to the user, and can be modified and saved.

We feel it is important to continue to extend these functions in an open-source (free) manner, since other options such as PARAXIA-Plus [2] or GLAD [3] cost in the thousands of dollars, while OPC is open source. In comparison to these existing programs, JLIFE not only dispenses with the need to construct a script (unlike GLAD) but also includes 3D graphing capabilities (unlike PARAXIA-Plus).

This paper presents details on the construction of the code for JLIFE, along with examples of results generated from using this program.

JAVA INTERFACE

We chose to implement this software in Java for two essential reasons. The primary rationale is that Java is an intrinsically cross-platform language, and thus our program can be run on any operating system that also supports Java. Secondly, Java has native libraries for constructing a Graphical User Interface (GUI) that is adaptable to many sizes of screens and to the look-and-feel for each platform. The back-end portion of code we elected to continue writing in Perl, the same scripting language used by OPC. Perl is a language highly suitable for quick text manipulation, which allowed us to offload memory-heavy in/out operations from Java onto Perl.

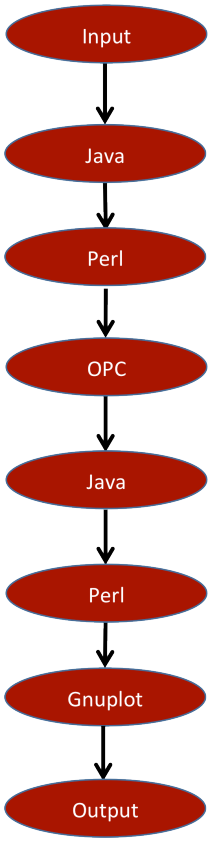

Figure 1: A schematic showing the control flow that JLIFE implements to handle each distinct function.

*Authored by Jefferson Science Associates, LLC and supported by the ONR, the Joint Technology Office, and the DOE under U.S. DOE Contract No. DE-AC05-06OR23177
#awatson@jlab.org

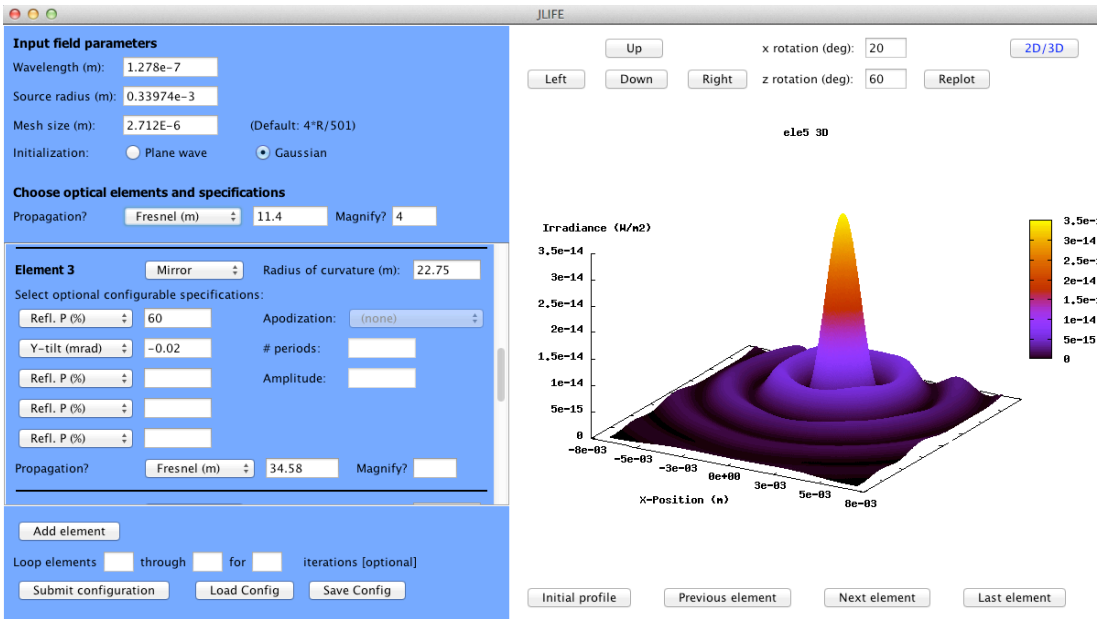

Figure 2: A screenshot of the graphical front-end of JLIFE. On the left-hand panel, users enter their desired specifications for an optical system. Field information and optical components are detailed in the same manner as in an OPC script. The user then selects the 'Submit Configuration' button to run the simulation. A graph showing the end result is displayed on the right-hand side upon completion of the calculation. Additional modifications of this view can then be made, including displaying profiles for element in the system and various 3D graphing capabilities.

The graphical output is produced using Gnuplot, a portable script-driven graphing utility created for the visualization of mathematical functions and other data [4]. Various types of 2D and 3D plots are constructed and saved as Portable Network Graphic (PNG) files, to continue utilizing free and portable data formats.

JLIFE first takes user input for the parameters of the optical system to be simulated (see Figure 2). Objects are constructed for each element that includes details such as the element's size or transmissive properties. These objects are then used to generate a Perl file in the proper format for compatibility with OPC. Care is taken to minimize the run time by utilizing a pre-written Perl script that acts as a general template applicable to any optical system. Thus it handles pre- and post-computation tasks such as locating the OPC libraries and cleaning up temporary files. The Java portion of JLIFE then executes the Perl script as an external runtime process and waits for confirmation that OPC has completed its execution.

A second Perl program is then externally executed to generate script files that contain commands for Gnuplot. This Perl script also calculates any necessary scaling factors due to user-specified magnifications or mesh sizes, and passes those values on to Gnuplot. Gnuplot then gathers the mode profile information from OPC data files, scales the data appropriately, and outputs the graphs in a PNG format. Control is returned to the main Java portion of JLIFE, which is then tasked with handling any view changes. Only a single graph is shown at a time; the user can select the output from the initial propagation, subsequent elements, or the final beam profile from among the buttons below the displayed graph. Each graph can be saved to a place of the user's choosing by simply right-clicking on the picture.

The user can also switch between viewing the default 2D cross-section or a 3D graph of the beam profile. Once the 3D view is selected, additional rotational capabilities are enabled. The view can be rotated in the z-direction (in and out of the plane) or in the x-direction (around the z axis). The replot button executes a Perl script to calculate the changes and re-write the input file for Gnuplot. Gnuplot is then called to re-plot just the modified 3D image, whereupon control is returned to JLIFE which re-displays that graph with the revolved view. Figure 3 exhibits the various styles of graphical output that JLIFE offers, including the contour map achieved by rotating a 3D image by 90 degrees out of the plane.

Instead of requiring a user to re-input their optical system each time JLIFE is used, save and load functions were written to ensure continuity of a configuration. Only the input information is saved, in a simple text file, which can be edited away from JLIFE or left to be loaded into JLIFE for the next time. No output graphs are saved unless explicitly specified by the user.

CONCLUSIONS

In this paper we have discussed the programming details employed in the new JLIFE software. This GUI effectively wraps the existing extensive capabilities of OPC in a way that frees the user from learning how to write Perl scripts or deal with plotting software. It is our expectation that this will make OPC more widely adopted by a greater number of scientific users for their optical calculations. In contrast to expensive software packages like PARAXIA-Plus and GLAD, this Java program is

provided free of charge as an additional open-source tool for modeling laser systems. Proposed future developments will focus on adapting JLIFE to any subsequent OPC versions and other FEL oscillator modeling programs. Execution of JLIFE could also be accelerated using a parallelized approach (MPI).

JLIFE is available for distribution now via an online download, which includes a readme file for all supported operating systems (Mac OS X, Windows XP or later, and most Linux flavors) and an installer package for Windows. A User's Manual is in preparation. Along with the Java executable, a test configuration is provided that demonstrates propagating a plane wave through an apodized aperture and subsequently focused using a lens.

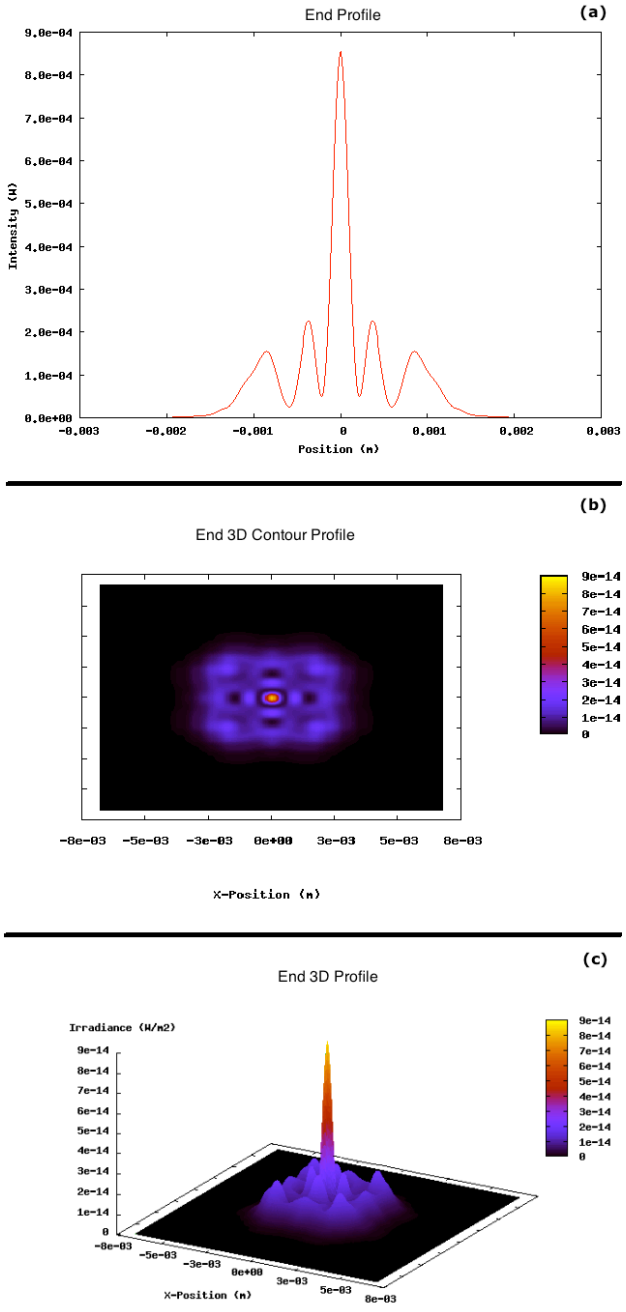

Figure 3: An example of (a) the default 2D beam profile, (b) the corresponding colored contour map, and (c) the rotated 3D graph.

ACKNOWLEDGMENTS

The authors would like to thank Stephen Benson for useful discussions, Peter van der Slot, one of the creators of OPC, for his close cooperation with us, Dominic Yurk for his coding work that added flexibility to the 3D plotting during a DOE-sponsored Summer High School internship at Jefferson Lab, and George Neil for proposing the acronym JLIFE.

REFERENCES

[1] J. G. Karssenberg, P. J. M. van der Slot, I. V. Volokhine, J. W. J. Verschuur, and K.-J. Boller, "Modeling paraxial wave propagation in free-electron laser oscillators", JAP 100, 093106 (2006).

[2] PARAXIA-Plus, Sciopt Enterprises, San Jose, CA 95120, USA.

[3] GLAD, Applied Optics Research, Woodland, WA 98674, USA.

[4] Gnuplot; www.gnuplot.info