Supplemental Document

Optics EXPRESS

Modeling of an X-ray grating-based imaging interferometer using ray tracing: supplement

JEFFREY P. WILDE[1,*] AND LAMBERTUS HESSELINK[1,2]

[1]E. L. Ginzton Laboratory, Stanford University, Stanford, CA 94305, USA
[2]Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA
[*]jpwilde@stanford.edu

Supplement DOI: https://doi.org/10.6084/m9.figshare.12727010

Parent Article DOI: https://doi.org/10.1364/OE.400640

Modeling of an X-ray grating-based imaging interferometer using ray tracing: supplemental document

We used a commercially available ray tracing software package, Zemax OpticStudio® [1], for our simulation work presented in the paper. This supplemental document describes some of the more salient details related to our specific version of the modeling implementation. However, it should become obvious that there exists significant flexibility to modify the model setup and parameters to best match a particular problem or geometry of interest. The discussion here assumes the reader has a basic working knowledge of OpticStudio. For more detailed information, the reader is directed to the comprehensive built-in help documentation provided with the software.

1. OVERVIEW OF THE ZEMAX MODEL (WITH A MATLAB INTERFACE)

The simulation runs in purely non-sequential mode. We use the Zemax OpticStudio Application Programming Interface (ZOS-API) in conjunction with Matlab [2] to write custom control and analysis code. ZOS-API is a very powerful object-oriented interface that allows external code to fully control the OpticStudio model setup and ray trace execution, so virtually any change that can be made via the graphical user interface inside OpticStudio can also be implemented programmatically. In addition, ray trace results can be stored to file and then imported into Matlab for post-processing. All of this can done with a single Matlab script file.

Various sources are available in OpticStudio, but we generally choose to use the "Source Two Angle" object with a circular spatial shape and a rectangular angular distribution of rays (chosen to optimally fill a rectangular detector). Source rays are chosen quasi-randomly by selecting the Sobol ray sampling method. They have a uniform distribution both spatially and angularly. The same set of source rays can be reproduced during repeated ray traces by setting the random number generator seed in OpticStudio to a fixed value using the following ZOS-API command: "TheSystem.Tools.OpenNSCRayTrace().SetRandomSeed(x)" where "x" is the seed value. The source spectrum is represented by a weighted set of wavelengths as shown in Fig. 3(b) of the paper (using either system wavelengths or an SPCD file).

Native 2D surfaces and/or 3D objects may be used. Externally generated CAD objects (e.g., in the form of STEP files) may also be utilized in OpticStudio. For volume objects, the refractive index and linear attenuation coefficient for a given material are included by means of a custom Zemax Table Glass (ZTG) file, which is a simple text listing in which each line consists of a wavelength in microns followed by the corresponding index and transmission data. We opt to use the specific source wavelengths in this ZTG file to avoid potential errors related to cubic spline fitting of the material values if a different wavelength sampling were to be used. Bulk scattering can be applied to solids, while surface scattering can be used on stand-alone surfaces or on the faces of volume objects.

We choose to turn off ray splitting, so each traced ray consists of just one branch having multiple segments from the source origination point to the final ray termination point. Each new segment arises at a ray-surface intercept or, in the case of volume scattering, at a ray scattering event in space. When ray splitting is off, no child rays are spawned, which helps keep the size of the ray data set manageable, and more importantly it maintains a one-to-one relationship between detected object rays and the corresponding reference rays. This helps simplify ray parsing and sorting when comparing the reference and object ray data sets. By not employing ray splitting we are neglecting spurious Fresnel reflection, which is quite reasonable in the X-ray regime given the very small refractive index decrements of typical materials. Also, when simulating surface scatter, it is in general possible to have many child rays split from one incoming parent ray; however, with the splitting function turned off, only one outgoing ray (either scattered or specular) leaves for each incoming ray. This approach is perfectly fine because tracing a large number of rays still leads to the proper statistical result.

In general, the coherent scattering cross-section is a function of energy (wavelength) and angle. However, the built-in bulk "Angle Scattering" distribution in OpticStudio is neither wavelength nor angle dependent. If that level of detail is important for a particular problem, then the best

solution would entail writing a custom scattering DLL (dynamic link library) as explained in the help documentation. Nevertheless, use of an average angular distribution is still quite useful for assessing the general impact of scattering on the visibility-contrast image.

2. RAY DATA SELECTION AND RECORDING

In OpticStudio, ray data sets can only be stored to disk and then read back for subsequent processing. One option, and perhaps the only approach that seems workable at first glance, is to use the standard Zemax Ray Database (ZRD) file format. Data are recorded for all segments of every ray, including the ray segment spatial coordinates, direction cosines, wavelength and intensity at each point where the segment intersects an object or is scattered in a new direction. For ray-object intercepts, the object number and face number are also stored. When a segment strikes a detector, the pixel number is included. While having access to this myriad of segment data can be helpful, it can pose a problem when scaling the model to include a large number of small objects, or especially when bulk scattering is turned on to simulate the visibility-contrast (dark-field) signal. The problem centers on the fact that every time a ray strikes an object boundary or is scattered, a new segment is created and the data values associated with that segment are added to the stored file. The number of ray segments can therefore grow exponentially. This in turn means that the ZRD ray data files may become excessively large, consuming more disk spacing and taking longer to write and to subsequently read and process. For our purposes, we only need data for the first and last ray segments, and possibly for the segment that intercepts G_1, but the rest of the segment data is superfluous.

We therefore devised a scheme to modify the method by which ray information is stored so as to better streamline the process. Fortunately, OpticStudio offers one other file format (SDF) for storing only those ray segments that strike a single user-selected object. After investigating this option more carefully, we discovered a way to make it work for our needs. We did this by adding one or two dummy rectangular volumes containing air to the model and instructing OpticStudio to only store ray segments that intersect the dummy volume(s). If objects are only placed in the beam path after G_1, then just one dummy volume is used with an input face located very close to the source plane (but the source is just outside of the dummy volume), and the opposing face of the dummy volume is very close to the detector surface (but the detector is positioned outside). Here very close means less than about 10 microns, although the exact value is not critical. The size of this dummy volume is large enough to enclose all of the objects in the beam path. Now when rays are traced, only the segments hitting the input and output faces of this dummy volume object are recorded to file, independent of the number of intervening segments generated during propagation. If objects are placed in the beam prior to G_1, then two dummy volumes are used as illustrated in Figure S1. The first volume has its input face adjacent to the source plane and

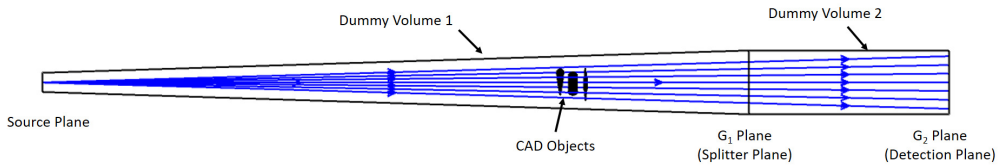

Fig. S1. Interferometer imaging model layout in Zemax OpticStudio.

its output face adjacent to the input side of G_1. The second dummy volume has its input/ouput faces adjacent to the output surface of G_1 and the detector plane, respectively. The two volumes are combined into a single dummy object by using a "Boolean Native" object. This allows the segments associated with the intersection of G_1 to be recorded as required by Eq. 19 in the paper. Actually, two intermediate segments are recorded, one on each side of G_1. Only one of these is needed and the second one is ignored. Finally, a filter string can be applied during ray tracing so that only the segments for rays that hit the detector are saved. Figure S2 is a screen shot of the ray trace control panel that illustrates an example of how this is accomplished.

Having just explained why the SDF file format is superior for capturing only the ray segments of interest, in simple cases one may opt for the ZDF format given the built-in ray database viewer tool for examining the contents of ZDF files. So, another important aspect related to non-sequential ray tracing in OpticStudio and storing the results in a ZRD file is worth highlighting. Ray tracing can be multi-threaded for a significant speed improvement, but the results are written

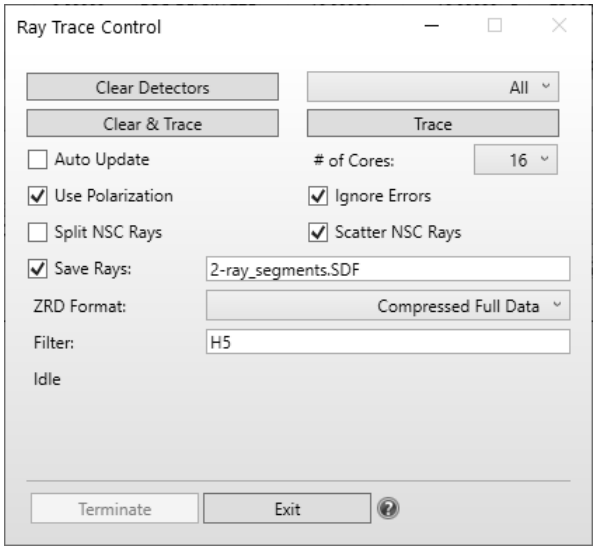

Fig. S2. Screen capture of ray trace settings window showing how ray segments that intersect dummy object number 2 are saved to an SDF file. Only segments for rays reaching the detector object number 5 are saved by using the H5 filter string.

to disk in the order that various threads complete execution. Therefore the ray number stored in the ZRD file cannot be used to compare the reference and object data sets. In other words, a particular source ray in the reference case will have an assigned ray number in the ZRD file, but the same physical source ray will, in general, have a different ray number in the object data set. Even for single-core ray tracing, the number of object rays may not correspond to the number of reference rays due to, for example, complete attenuation of object rays by metallic objects. For these reasons, even when working with ZRD files, we still use the first ray segment coordinates and direction cosines for purposes of sorting in Matlab.

3. RAY DATA PARSING IN MATLAB

After ray tracing is complete, the Matlab script includes instructions to read the SDF files from disk (one file for the reference ray trace and a second file for the object trace) and to then separate the reference and object ray segments into three groups: the first group corresponds to the source segments, the second is for segments incident on G_1, and the third is for segments incident on the detector (or G_2 plane). Note that the reference segment groups all have the same number of elements, as do the object segment groups, but the object groups may have a smaller number of elements compared to the reference groups. This is because during the object ray trace, some of the rays may be completely attenuated by an object or there could be a ray trace error associated with a ray striking the facet boundary of a CAD object. If an object ray does not make it all the way to the detector, then the filter string prevents that ray (and its segments) from being saved. Knowing this could be the case, the first group of ray segments (i.e., those emitted by the source) for both the reference trace and object trace are compared and sorted (based on spatial coordinates and direction cosines) using the "intersect" command in Matlab. Doing so ensures that these initial segments for the two traces are ordered in precisely the same way and also eliminates any reference segments that have no corresponding object segments. The detector ray segments (and the G_1 intersection segments if needed) can then be sorted using the array index vectors generated by the "intersect" operation. The reference and object ray segment arrays can now be used on a segment-by-segment basis to complete the required analysis according to Section 4.4 of the paper.

4. SUMMARY

The most complicated aspect of the ray trace model as implemented in OpticStudio pertains to the ray parsing and sorting in Matlab, followed by binning the results to mimic a pixelated detector. However, the optical path construction in OpticStudio is quite straightforward, and the

multi-threaded ray tracing runs efficiently. To summarize, we use a Matlab script to control the OpticStudio simulation according to the following main steps:

1. First, select "Ignore" for all objects and run an air ray trace to serve as a reference. Save the ray segment data to file. Rays that do not reach the detector are not saved.
2. Turn the objects back on by deselecting "Ignore" and rerun the ray trace using exactly the same set of source rays (by using the same random number generator seed value and Sobol ray sampling). Save this object ray data to a second file. Again, rays that do not reach the detector are not saved.
3. Read the ray data sets from disk. Separate the ray segments into three groups (source segments, segments that intersect G_1, and segments that intersect G_2), and then compare and sort the segments within each group as described in Section 3 above.
4. Compute the difference in the x-direction ray crossing coordinates at G_1 and G_2 and find the local fringe phase shift for each reference/object ray pair using Eq. 19.
5. Evaluate the phasor sum per pixel according to Eq. 26.
6. Calculate image projections for the three modalities per Eqs. 32-34.

REFERENCES

1. OpticStudio 20.1, Zemax, LLC, Kirkland, Washington, United States (www.zemax.com).
2. MATLAB 2019b, The MathWorks, Inc., Natick, Massachusetts, United States (www.mathworks.com).