SIMPLEHEAT:
X-RAY HEAT LOADING FOR THERMAL MODELLING

Richard Hilliard, CHESS, Cornell University, Ithaca, NY 14853, U.S.A.

Abstract

Heat loading is a critical concern in x-ray optics. Deformations (heat bumps) may form on surfaces absorbing high energy, high power x-rays. These thermal bumps limit coherence and reduce flux in the beam, thus degrading the quality of the x-ray probe. SimpleHeat is a software package that models three-dimensional power absorption in beamline components due to beams from both undulators and wigglers. This data can then used to calculate thermal profiles and predict energy deposition and deformation.

MOTIVATION

Only a small portion of the total power in the x-ray beamline reaches the monochromator due to upstream filters and apertures, but this power can still cause considerable deformation in the first crystal, leading to decreased x-ray flux and deterioration of the tuned beam, limiting the coherence before diffraction.

The SimpleHeat software enables quick and accurate prediction of anticipated damages and deformations in variable objects and filter configurations due to synchrotron x-rays. The software is intended to minimize the time spent by engineers, technicians and scientists on empirical determination of heat-load modeling and provide accurate information to determine the necessary characteristics of cooling schemes for optical elements of the beamline.

ARCHITECTURE

The SimpleHeat software is written in Python 3, its graphic user interface (GUI) is implemented with Qt [1], using PyQt4 [2], a Python binding plug-in which enables the use of Qt's GUI modules from Python code. Additionally SimpleHeat utilizes XOP [3] to map radiation spectra from insertion magnets.

`heatloadmatrix.py`

The main script, 'heatloadmatrix.py', calls and organizes the various sub-scripts, sub-functions and procedures to perform the desired tasks during its execution. This task order is set through a series of switches interfaced through the GUI selecting for insertion device (and applicable parameters), object configuration, mesh, filter type and power output. SimpleHeat supports undulator and wiggler insertion devices and will incorporate bending magnet functionality in the future. To calculate the characteristic radiation spectra of wigglers and undulators, SimpleHeat invokes XOP.

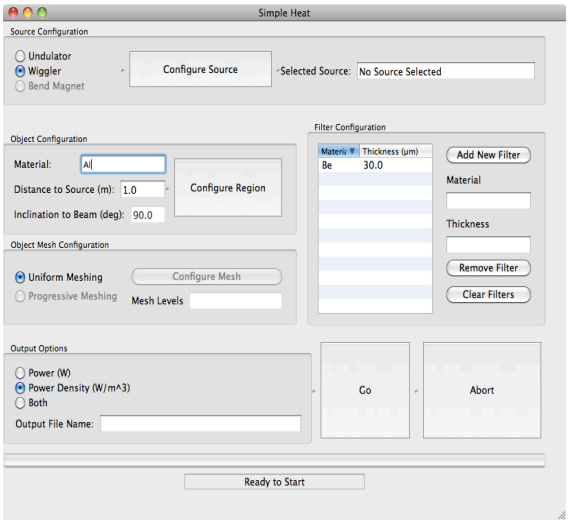

Figure 1: The SimpleHeat GUI, built with Qt.

XOP 2.3

SimpleHeat uses two executable files from XOP 2.3 to create the wiggler and undulator radiation spectra to develop spatially and spectrally resolved heat loads given specified synchrotron source parameters. XOP was developed at ESRF by Manuel Sanchez del Rio and Roger J. Dejus and was last patched in June of 2013 (installations instructions for XOP are included in the SimpleHeat documentation). Possible parameters include: beam energy, beam current, period, number of periods, deflection in the x direction, deflection in the y direction, minimum photon energy, maximum photon energy, number of energy steps, distance from source, x and y positions of the aperture, width (x) of slits, length (y) of slits and the number of integration points in the x and y directions. These two executable files, '*ws'* for wigglers and *'us'* for undulators, generate two-column tables ofphoton energy (for each specified energy step) against power per .01% bandwidth of the given energy.

At each energy step, the energy range is set up by the specified minimum and maximum photon energies.

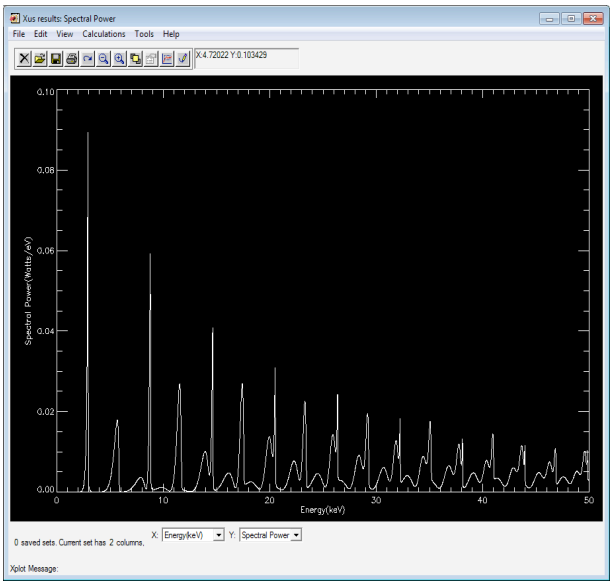

Figure 2: an XOP generated undulator spectrum.

Running either the wiggler or undulator executables of XOP generates a single radiation spectrum and power for an aperture of the specified fixed size and location relative to the centered of the beam described. SimpleHeat runs these executables, using small apertures, for each "brick" designated by the specified number of steps in the x and y directions until it has covered the target. This process generates a spectrum and a power (in Watts) for every location on the surface, including those outside of the beam footprint where the crystal is often distorted nonetheless.

backend_worker.py

The majority of the mathematical heavy lifting and miscellaneous classes/functions are implemented in this module. The class 'Back' imports all of the region and filter values, beam, machine and insertion device parameters which are initialized and stored to '.json' files in a project subdirectory. Processing modules are imported to decrease clock cycles. Ultimately, a subroutine checks that total power is equal to power absorbed plus power transmitted:

$$p = t + a$$

to ensure that there are no deep errors in computation and the output is accurate.

From the initialized and specified values at the GUI level, 'backend_worker' does the work of assembling the XOP formatted, two dimensional (x and y directions) matrices for wigglers or undulators, and then applies the specified depths to calculate the volumes and create the 'brick matrix' and finally runs XOP to generate the necessary spectrum at each stage for each 'brick'.

Subsequently, the transmissions of flux through each filter are calculated. First, the function 'filter_flux' finds the transmissions through one or more filters using Beer's Law:

$$I = I_o\, e^{-\alpha x}$$

where 'I' is the intensity of the beam which decreases as a function of 'x', the penetration depth. Alpha (α) is the mass attenuation coefficient of the given material. The total transmission reaching the target is the difference between total transmission minus the total intensity absorbed by each and every filter.

Absorption for each brick in the target object is calculated in a similar manner to flux transmission. From the total power reaching the surface, absorption in each brick is returned as power in Watts. For successive bricks, the relation is:

$$I = I_o\, e^{-\alpha x2} - I_o\, e^{-\alpha x1}$$

with the difference between the depths x2 and x1 signifying the depth or height of each brick, and the intensity value decremented through each iteration and checked against the total power equation mentioned above.

Ag, Al, Au, Be, Br, C, Co, Cu, Fe, Hg, Mn, Ni, Pb, Pt, Rb, Se, Si, Ta, Zn

Figure 3: filters and objects.

The alpha values (α), the mass attenuation coefficients, are interpolated within the user directed range from two-column .csv files in the 'mu_data' directory which list empirically determined absorbances of each object material at a given energy level of photon. These values are taken from the National Institute of Standards and Technology website. To date, SimpleHeat supports 19 filter and object materials [see Figure 3s], with beryllium, carbon and silicon being the most commonly used at CHESS; the former in entrance windows (visible in Figure 4) and the latter two as monochromators. The default parameters in SimpleHeat

match those used in the CHESS West A2 hutch setup. To use some other filter or object, the user simply needs to create a new 'mu_data' spreadsheet from NIST values or another resource, appropriately name the file and call it from the GUI.

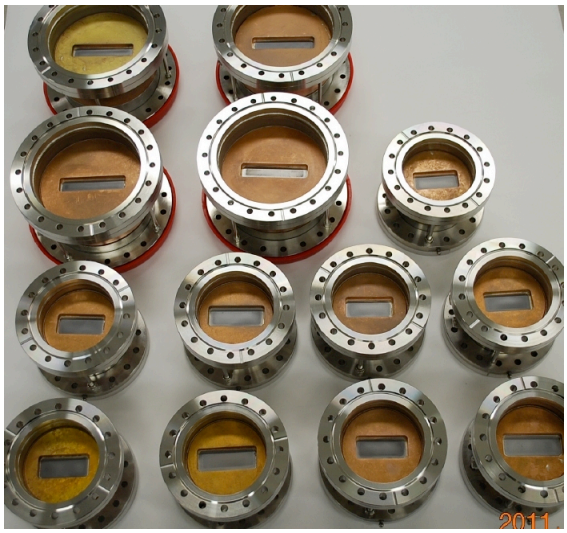

Figure 4: beryllium entrance windows produced by Advanced Design Consulting.

Attenuation through a filter alters the radiation spectrum. After each successive filter, the spectrum is redrawn along each photon energy and returned for the next calculation. The ease of altering, adding and subtracting filters and objects to run against the same machine parameters is a principal mechanism by which SimpleHeat reduces x-ray community work-hours spent on experimental setup and testing of different components for heat loading.

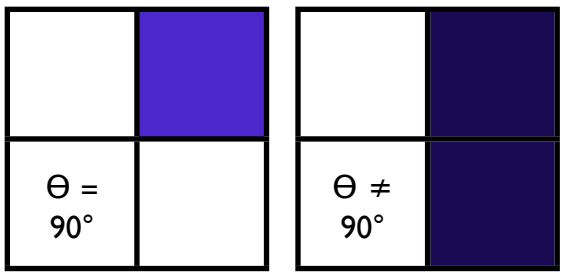

Figure 5: Projection and mirroring scheme set by incident angle.

Lastly, the flux load (photon energy) is translated to power and power is divided by volume to obtain power density. A switch in the GUI enables one or both of these variables as output. The power (Watts) and power density (Watts per cubic meter) of each brick are written as output to tables with their corresponding coordinates in the x, y and z-directions.

Depending on the inclination of the beam, SimpleHeat calculates either a one-quarter or a one-half projection and then mirrors those values. This is done to minimize computation and makes the reasonable assumption of beam symmetry about the vertical axis. For a perpendicular beam, a quarter projection made and then mirrored three times; for a beam of any angle other than 90º, a half projection is made and mirrored once about the vertical axis.

FIXES

Multiple fixes were made to bugs carried over from the last (2012) version of the program, formerly 'HeatBump' [4]. These are listed in detail in the packaged program files. The default and user-set parameter files (.json) were frequently corrupted, causing the program to crash. This was fixed by changing a number of call orders, file extensions and switches for read/write parameters. The GUI windows were reformatted for greater readability using the Qt designer tool. A number of Unicode decoding errors were fixed within the XOP calls. The region function was fixed to correctly separate and assign the brick thicknesses or 'slice' values. The .ui files for the GUI, which are built and read by Qt, were updated and rerun through the 'pyuic4' script to create the correct .py files.

An OS X version was developed using the same Python 3.3.2, Qt 4.8.4, and PyQt4.10.2 configuration that the Windows 7 development version used. The OS X version additionally uses the GNU Fortran distribution for the front-end compiler and GCC libraries. The distribution presently includes two versions, one for each operating system, but it should be possible to engineer the distribution to provide a build capable of running on either system.

DOCUMENTATION

SimpleHeat is outlined and characterized in the SimpleHeatWiki, that will be hosted on the CHESS website. The logged changes, version history and directives for the future are contained within the packaged SimpleHeat software. The SimpleHeatWiki contains a Quick Start Guide, a Program Guide describing more complex operations, an About section, public licensing details as well as installation instructions.

The installation and building of the SimpleHeat software and all of the requisite tools are nontrivial tasks

for the novice computer user, so special attention was paid to their precise documentation. Packaged binary installers exist for the Windows machines, simplifying much of this process. However, the Macintosh installation instructions should be followed closely as the needed software must be installed and built from the terminal.

```
source_flux s_flux = 9.772539657693379e+17
integrated_source_power s_flux = 9.772539657693379e+17
integrated_source_power intpow = 955.6021102206761
filter_flux s_flux = 9.772539657693379e+17
filter_flux f_flux layer(1) = 3232590765023.9614
filter_flux f_flux layer(2) = 6.459689131165875e+17
filter_flux f_flux layer(3) = 61095351827.76814
filter_flux f_flux layer(4) = 4.5787120526403425e-06
filter_flux f_flux layer(5) = 7.037062163372102e-46
filter_flux f_flux layer(6) = 5.8411644028923426e-123
filter_flux f_flux layer(7) = 6.601083659742032e-250
filter_flux f_flux layer(8) = 0.0
filter_flux f_flux layer(9) = 0.0
filter_flux f_flux layer(10) = 0.0
filter_flux f_flux layer(11) = 0.0
filter_flux f_flux layer(12) = 0.0
integrated_source_power s_flux = 6.459722068027044e+17
integrated_source_power intpow = 891.1722626660293
region_filtered_flux s_flux = 6.459722068027044e+17
region_filtered_flux reg_flux = 6.459722068027044e+17
slice_transmission slice_t layer(1) = 3583.391262769676
voxel_absorbed_flux s_flux = 6.459722068027044e+17
voxel_absorbed_flux slice_t = 3583.391262769676
voxel_absorbed_flux slice_absorption = 1416.6087372303152
voxel_absorbed_flux cumulative_transmission layer(1) = 5000
voxel_absorbed_flux voxel_impinging_flux layer(1) = 6.459722068027044e+17
voxel_absorbed_flux voxel_absorbed_flux layer(1) = 5.947825791910319e+17
voxel_flux_to_power v_flux layer(1) = 5.947825791910319e+17
voxel_flux_to_power int_v_power layer(1) = 653.2877031094783
voxel_absorbed_power_density v_power layer(1) = 653.2877031094783
voxel_absorbed_power_density slice_volumes layer(1) = 1e-07
voxel_absorbed_power_density v_power_density layer(1) = 7603233955810.733


total_integrated_power v_power layer(1) = 653.2877031094783
total_integrated_power returns = 653.2822540849833
integrated_source_power s_flux = 5.1189626518020 7e+16
integrated_source_power intpow = 237.89000525190983
Title: toot
Integrated source power without filtering: 955.6021102206761 W
Integrated source power after filtering: 891.1722626660293 W
Integrated power absorbed in the object: 653.2822540849833 W
Integrated power transmitted through the object: 237.89000525190983 W
Integrated power error: 3.32913617739905e-06 W
XOP/Flux time: 59.4155158996582 s
Processing time: 12.835970163345337 s
Elapsed time: 72.25148606300354 s
rrdhcp-10-32-53-79:simple rfh$
```

Figure 6: The terminal output from a SimpleHeat run.

OUTPUT & ANALYSIS

After 'heatloadmatrix.py' runs, output .csv files are created for power (Watts) and/or power density (Watts per cubic meter) mapped to the matrix of bricks. These files are then used to calculate thermal profiles and predict deformation of optics. A finite element modeler (typically, ANSYS at CHESS) is then used to turn the power values and brick matrix to a 3-dimensional model of the heat bump on the component optical crystal using a steady-state thermal distribution and a static structural simulation[5].

CONCLUSION

Tests have found that the slope error profile of these heat load models very closely approximate the experimentally measured slope error profiles in head loaded optics [6]. SimpleHeat promises to help scientists at both CHESS and the broader x-ray community determine which optics will best function at specific beamlines, and what cooling schemes will be necessary to mitigate heat load for those optics.

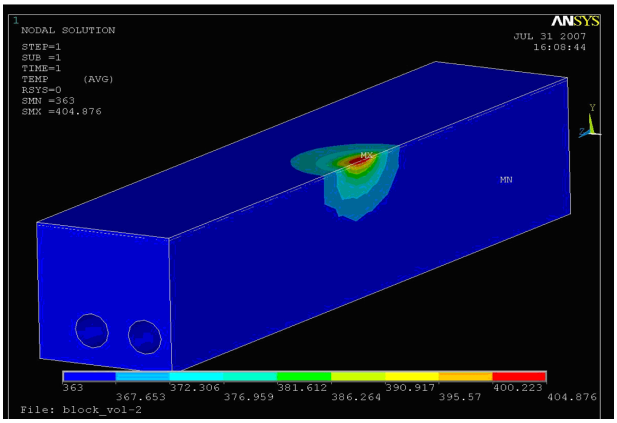

Figure 7: A simulated 'heat bump' modeled by ANSYS .

ACKNOWLEDGMENTS

This project would not have been possible without the direction and support Jim Savino and Aaron Lyndaker. Additional gratitude is directed towards the many others at CHESS, both past and present, who have worked on and around the development of this project. Thanks go also to Matthias Liepe who has championed this project and others like it. This work is supported by the National Science Foundation under Grant No. 0841213.

REFERENCES

[1] Qt, https://qt-project.org/', (2013).
[2] PyQt, http://www.riverbankcomputing.com/software/pyqt/intro, (2013).
[3] XOP, http://www.esrf.eu/Instrumentation/software/data-analysis/xop2.3, (2013).
[4] Hilliard, 'SimpleHeatWiki,' CHESS website, (2013).
[5] Revesz, et al., 'Heat-bump Measurements at CHESS A2 Wiggler Beam,' CHESS News Magazine, (2009).
[6] Revesz, et al., 'Heat-bump Measurements at CHESS A2 Wiggler Beam,' CHESS News Magazine, (2009).

FIGURES

Figure 8:
SimpleHeat module
dependency chart.