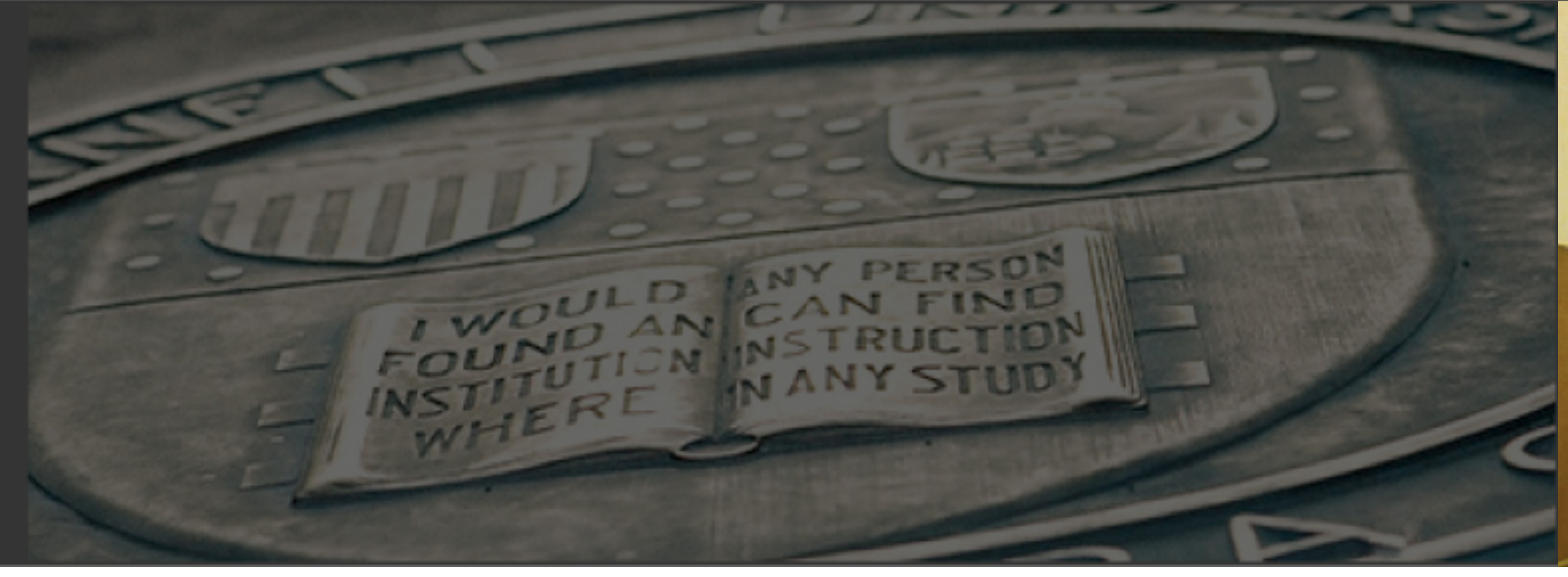
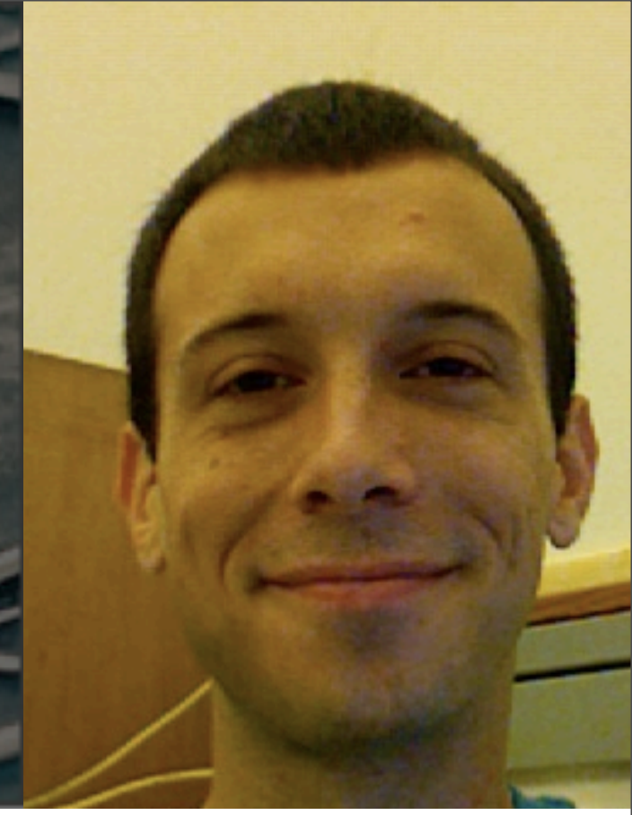
Cornell University
Laboratory for Elementary-Particle Physics

Rich Hilliard
Tompkins-Cortland Community College
Jim Savino, Aaron Lyndaker

SR CCS 2013


I WOULD FOUND AN INSTITUTION WHERE ANY PERSON CAN FIND INSTRUCTION IN ANY STUDY

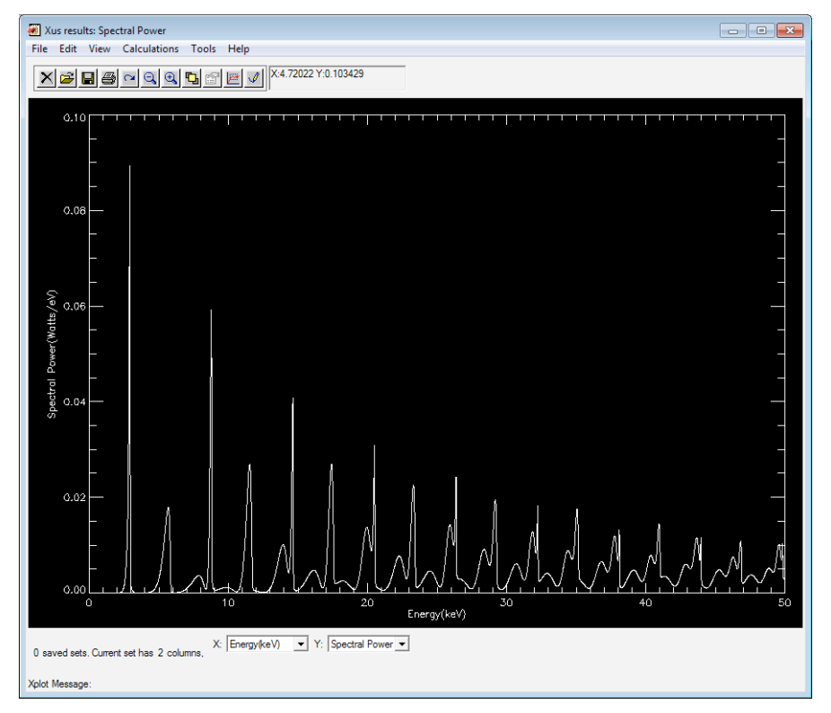


Simple Heat: X-Ray Heat Loading for Thermal Modeling

Summer Research for Community College Students - 2013

Simple Heat

Simple Heat is a software package that models three dimensional power absorption in beamline components due to high energy, high power X-ray beams from both undulators and wigglers. This data is then used to calculate thermal profiles and predict deformation of optics.

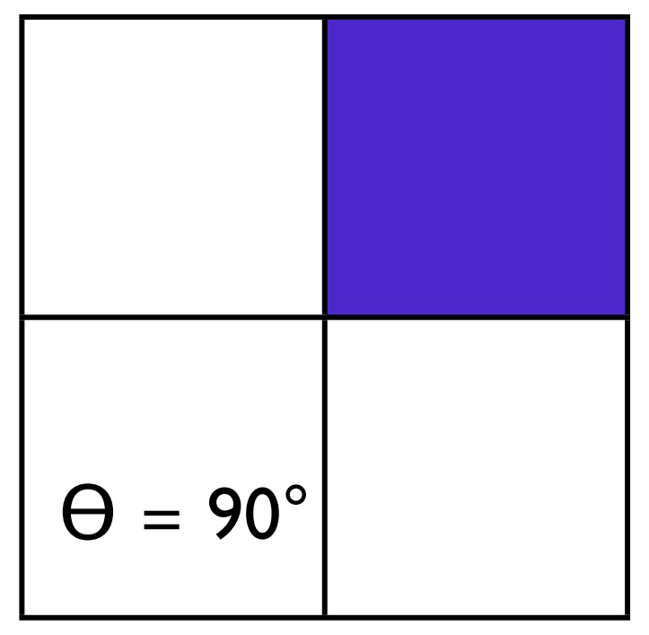


Operation Simple Heat takes in values for numerous beam and synchrotron parameters and generates a characteristic X-ray spectrum (right: an undulator spectrum).

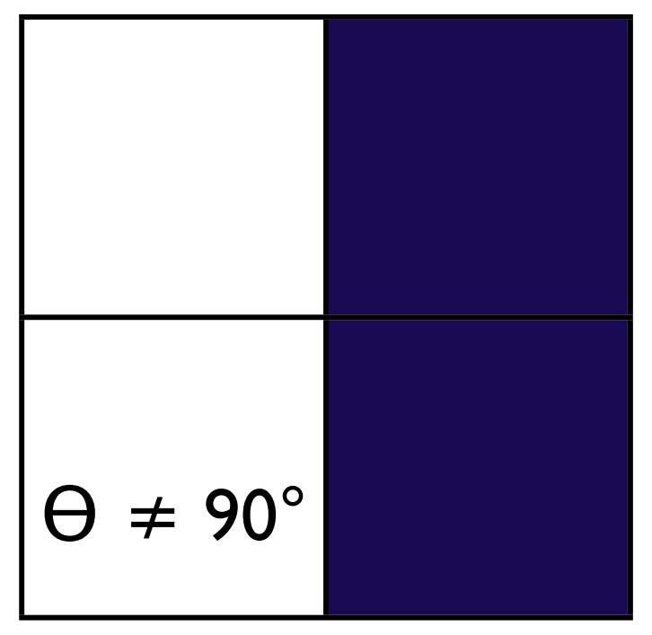
XOP Simple Heat uses XOP (developed at ESRF) to develop spatially and spectrally resolved loads.

Energy absorption Values for local heat/energy absorption are calculated in 3 dimensions and translated to a brick matrix (brick powers and power densities are output to a file).

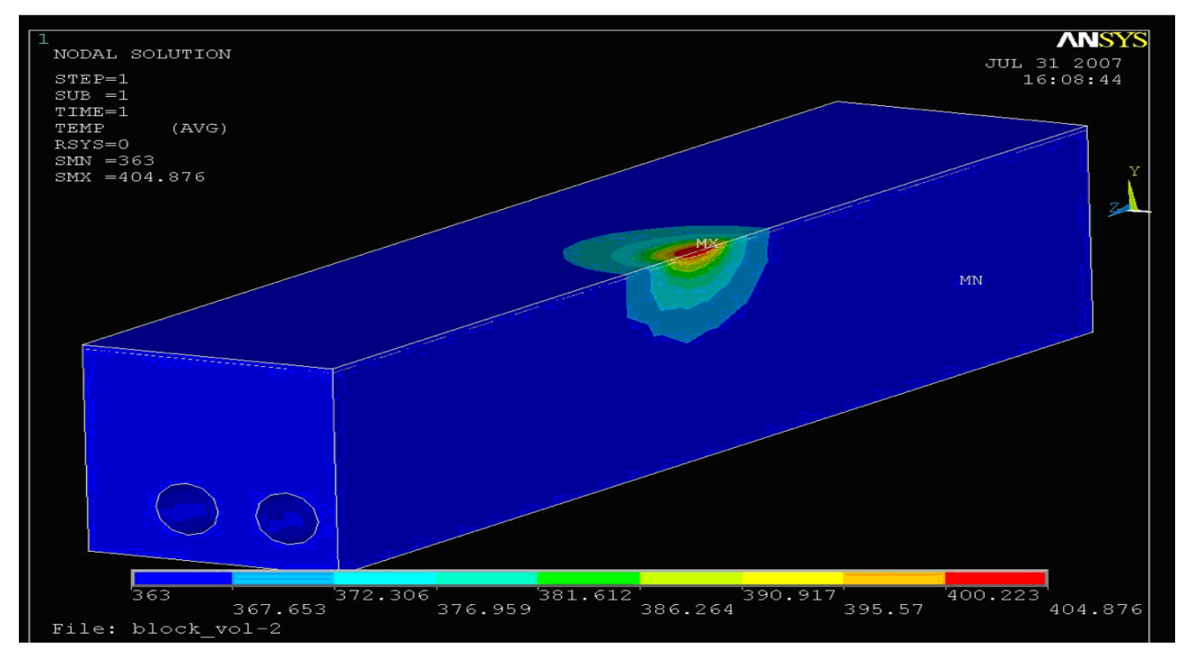
Projection The beam is assumed to be symmetrical about the vertical axis. If the incident angle, theta, is 90° then a quarter projection is mirrored three times. If theta is not 90° then half of the beam is computed and mirrored once.



Θ = 90°



Θ ≠ 90°



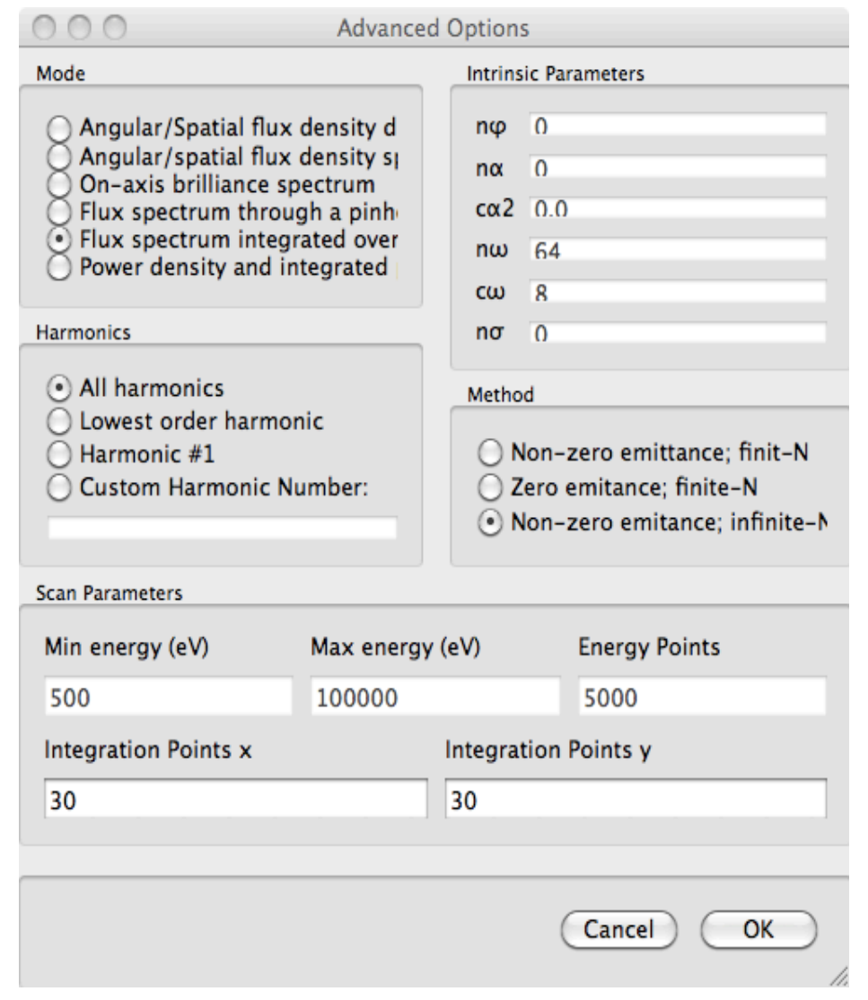
Output The mapped loads can then be used by a Finite Element Modeler (FEM), e.g. ANSYS, to calculate the bump. (above right, an ANYSYS model of a heat bump)

Motivation

Heat load modeling allows for accurate predictions of the damage and deformation that X-rays will inflict on optics and other highly loaded components. Given this information, it is then possible to make quick and cost-effective decisions about the temperature control components necessary to the maintain quality at the interface of beam and equipment.

GUI

Parameters



Advanced Options

Mode
○ Angular/Spatial flux density d
○ Angular/spatial flux density s
○ On-axis brilliance spectrum
○ Flux spectrum through a pinh
● Flux spectrum integrated over
○ Power density and integrated

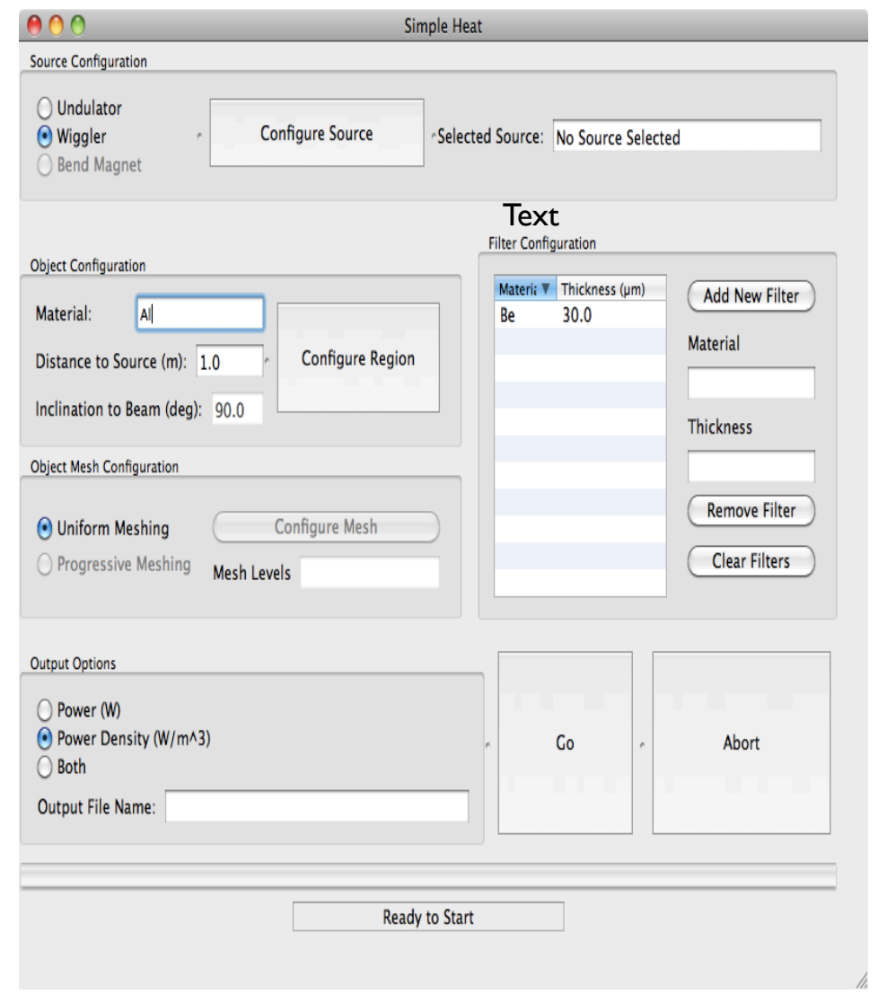
Intrinsic Parameters
nφ 0
nα 0
cα2 0.0
nω 64
cω 8
nσ 0

Harmonics
● All harmonics
○ Lowest order harmonic
○ Harmonic #1
○ Custom Harmonic Number:

Method
○ Non-zero emittance; finit-N
○ Zero emitance; finite-N
● Non-zero emitance; infinite-N

Scan Parameters
Min energy (eV) 500
Max energy (eV) 100000
Energy Points 5000
Integration Points x 30
Integration Points y 30

Main Window



Simple Heat

Source Configuration
○ Undulator
● Wiggler
○ Bend Magnet
Selected Source: No Source Selected

Text
Filter Configuration

Material	Thickness (μm)
Be	30.0

Object Configuration
Material: Al
Distance to Source (m): 1.0
Inclination to Beam (deg): 90.0

Object Mesh Configuration
● Uniform Meshing
○ Progressive Meshing
Mesh Levels

Output Options
○ Power (W)
● Power Density (W/m^3)
○ Both
Output File Name:

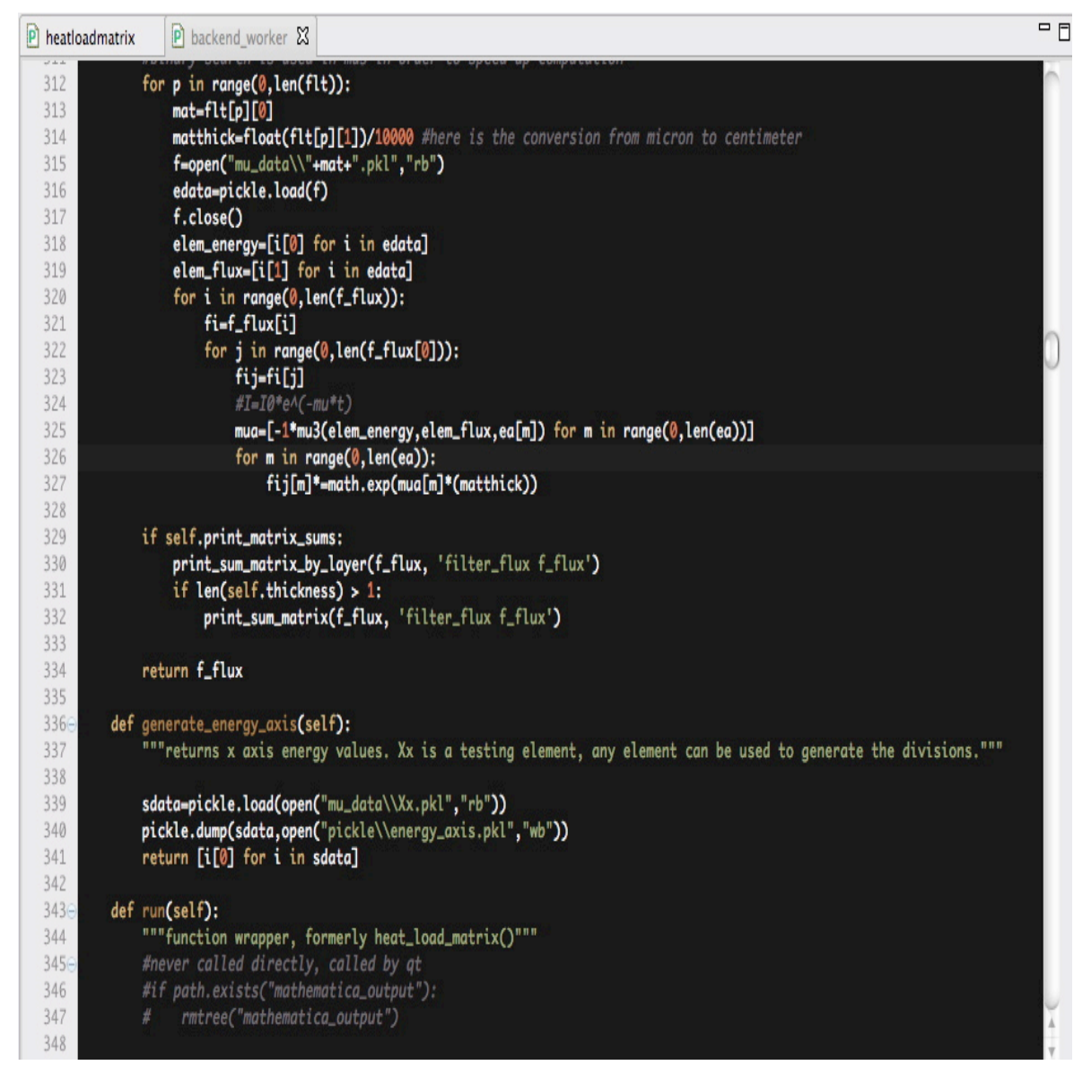
Ready to Start

Objects / Filters

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

Code

The primary script, heatloadmatrix.py, calls the necessary functions, imports modules and organizes all the dependents scripts. Output files are written to the .csv type. Parameters and object arrays may be saved to run later simulations and analyses with different objects, filters, machine and beam configurations. (right: a sample function and a workflow diagram).



heatloadmatrix | backend_worker

```
        for p in range(0,len(flt)):
            mat=flt[p][0]
            matthick=float(flt[p][1])/10000 #here is the conversion from micron to centimeter
            f=open("mu_data\\"+mat+".pkl","rb")
            edata=pickle.load(f)
            f.close()
            elem_energy=[i[0] for i in edata]
            elem_flux=[i[1] for i in edata]
            for i in range(0,len(f_flux)):
                fi=f_flux[i]
                for j in range(0,len(f_flux[0])):
                    fij=fi[j]
                    #I=I0*e^(-mu*t)
                    mua=[-1*mu3(elem_energy,elem_flux,ea[m]) for m in range(0,len(ea))]
                    for m in range(0,len(ea)):
                        fij[m]*=math.exp(mua[m]*(matthick))

        if self.print_matrix_sums:
            print_sum_matrix_by_layer(f_flux, 'filter_flux f_flux')
            if len(self.thickness) > 1:
                print_sum_matrix(f_flux, 'filter_flux f_flux')

        return f_flux

    def generate_energy_axis(self):
        """returns x axis energy values. Xx is a testing element, any element can be used to generate the divisions."""

        sdata=pickle.load(open("mu_data\\Xx.pkl","rb"))
        pickle.dump(sdata,open("pickle\\energy_axis.pkl","wb"))
        return [i[0] for i in sdata]

    def run(self):
        """function wrapper, formerly heat_load_matrix()"""
        #never called directly, called by qt
        #if path.exists("mathematica_output"):
        #    rmtree("mathematica_output")
```

i.d. / bending magnet	filter(s)	region
[XOP]	radiation spectrum	heat load
project	mirror	output

Goals

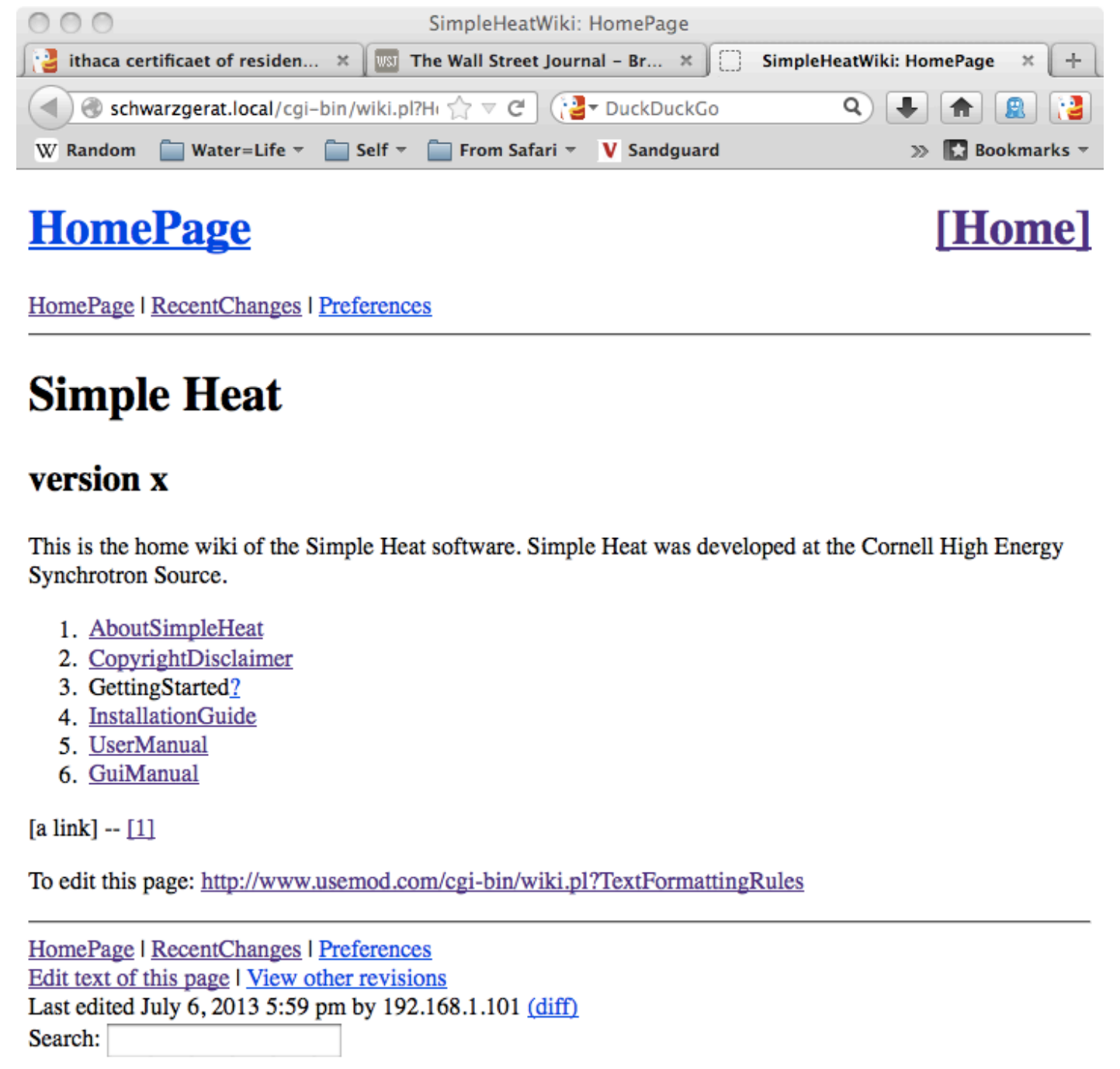
Debug, Document & Publish:
Make such heat load modeling easy to execute for members of the X-ray community.

Fixes

· Unicode decoding errors · File pathing improvements · Improved 'region' function · GUI readability · Universalized certain file paths · Fixed json file errors · Fixed corrupted input files · ran pyuic4 script to update .ui files · GNU Fortran (Mac version)

Errors

```
separate Read - Write files?grep -nH -e "\"w\"" *.pyadvancedoptions.py:
60:     f=open("pickle/adv.json","w")backend_worker.py:451:
f=open("heatload_results.txt","w")backend_worker.py:504:
f=open(root+"/"+name+".dat","w")backend_worker.py:518:
f=open(root+"/"+name+"_"+str(m)+".dat","w")backend_worker.py:523:
f2=open(root+"/"+name+"_"+str(m)+"unit.dat","w")backend_worker.py:
585:        f=open(jobdir+'/'+ xop_pgm + ".inp", "w")
backend_worker.py:1023:     f=open("raw_flux.csv","w")
backend_worker.py:1108:     f=open(outputfile,"w")heatloadmatrix.py:
109:     f = open(filename, "w")heatloadmatrix.py:129:
json.dump(source, open("pickle/wig.json", "w"), indent=2)
heatloadmatrix.py:130:        json.dump(source, open("pickle/
wigload.json", "w"), indent=2)heatloadmatrix.py:134:        json.dump
(source, open("pickle/und.json", "w"), indent=2)heatloadmatrix.py:135:
json.dump(source, open("pickle/undload.json", "w"), indent=2)
heatloadmatrix.py:155:     with open("pickle/flt.json", "w") as
f:heatloadmatrix.py:238:     json.dump(run, open("pickle/run.json",
"w"), indent=2)heatloadmatrix.py:271:     json.dump(json.load(open
("pickle/und.json", "r")), open("pickle/undload.json", "w"))
heatloadmatrix.py:272:
```



SimpleHeatWiki: HomePage

ithaca certificaet of residen...
The Wall Street Journal - Br...
SimpleHeatWiki: HomePage
schwarzgerat.local/cgi-bin/wiki.pl?H
DuckDuckGo
Random
Water=Life
Self
From Safari
Sandguard
Bookmarks

HomePage [Home]

HomePage | RecentChanges | Preferences

Simple Heat

version x

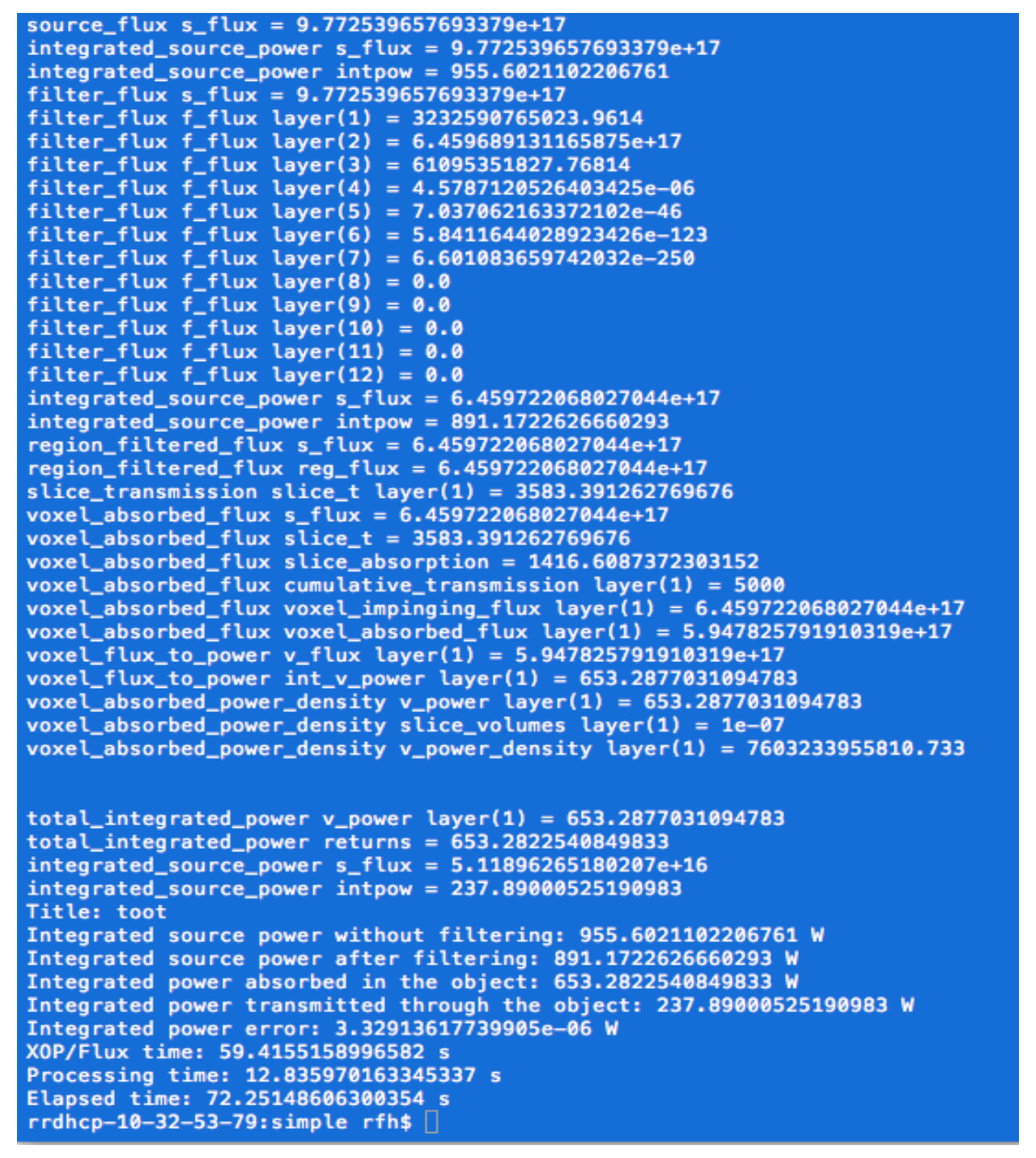
This is the home wiki of the Simple Heat software. Simple Heat was developed at the Cornell High Energy Synchrotron Source.

1. AboutSimpleHeat
2. CopyrightDisclaimer
3. GettingStarted?
4. InstallationGuide
5. UserManual
6. GuiManual

[a link] -- [1]

To edit this page: http://www.usemod.com/cgi-bin/wiki.pl?TextFormattingRules

HomePage | RecentChanges | Preferences
Edit text of this page | View other revisions
Last edited July 6, 2013 5:59 pm by 192.168.1.101 (diff)
Search:



```
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 = 5.11896265180207e+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$
```

Documentation

Hosted on a wiki (left)
· About
· Installation Instructions
· Operations Manual
· Quick Start Guide
· Parameter Restrictions

Publish

· Version History
· GNU Public License
· Packaged Directory

Acknowledgements
Jim Savino, Aaron Lyndaker, Rong Huang, Ben Oswald, Matthias Liepe

This work is supported by the National Science Foundation under Grant No. 0841213.
Any opinions, findings, and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.