

# Novel, Fast, Open-Source Code for Synchrotron Radiation Computation on Arbitrary 3D Geometries

Dean Andrew Hidas

**BROOKHAVEN**  
NATIONAL LABORATORY

 U.S. DEPARTMENT OF  
**ENERGY**

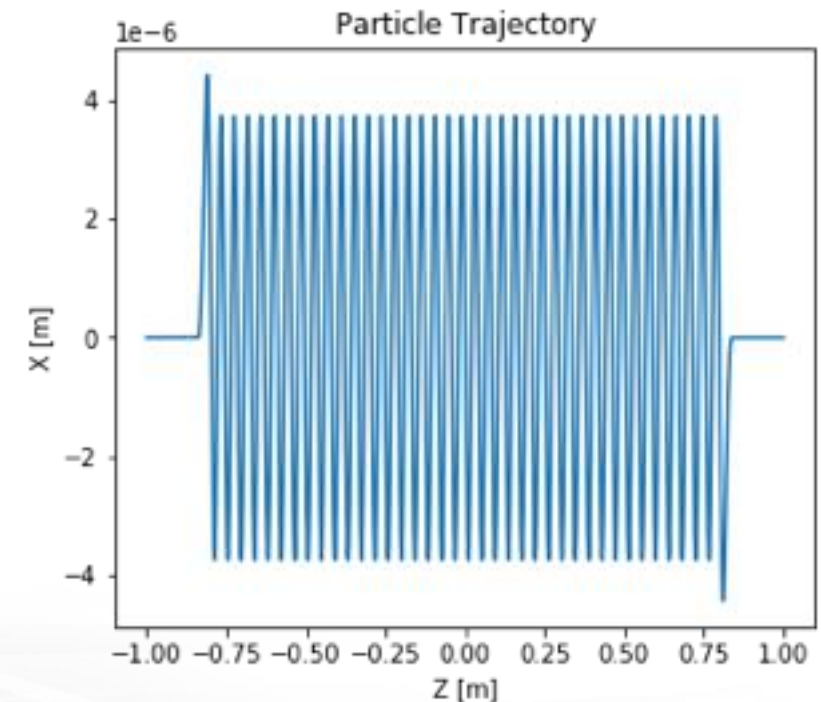
# Outline

- Synchrotron Radiation Basics
- Introduction to OSCARS
- Basic SR Calculations
- Calculations on 3D Geometries
- Time Dependent E, B

# SR Basics

- Goal of looking at a beam and calculating radiative properties
  - Beam, Bending Magnets, Quads, Insertion Devices, Arbitrary Fields
- First is to solve for the trajectory
  - 2<sup>nd</sup> order differential equation
  - Typically solved via 4<sup>th</sup> order Runge-Kutta method

$$\frac{d\vec{p}}{dt} = q(\vec{E} + \vec{\beta}c \times \vec{B})$$



# SR Basics

- Calculate a quantity of interest (from trajectory information)

- Flux, from E-field

$$\vec{E}(\vec{x}, \omega) = C \int_{-\infty}^{+\infty} \left[ \frac{1}{\gamma^2} \frac{\hat{n} - \vec{\beta}}{R^2 (1 - \hat{n} \cdot \vec{\beta})^2} + \frac{\hat{n} \times (\hat{n} - \vec{\beta}) \times \dot{\vec{\beta}}}{R (1 - \hat{n} \cdot \vec{\beta})^2} \right] e^{i\omega(\tau + R/c)} d\tau$$

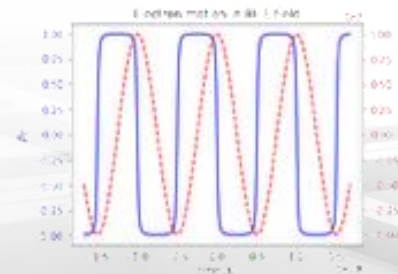
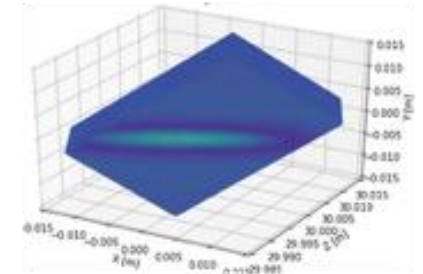
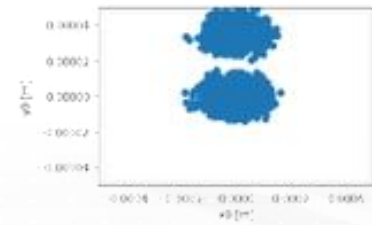
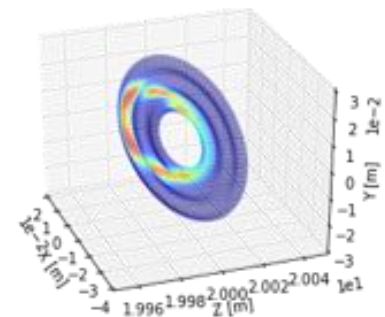
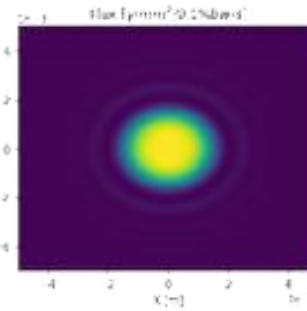
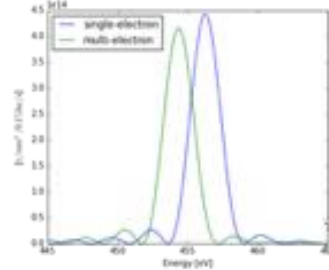
- Numerically integrate this (and the like) to desired accuracy
  - Smaller trajectory steps = higher accuracy, cost is time and memory
  - GPU and multi-threading good candidates for this



# Introduction to OSCARS

- Open Source Code for Advanced Radiation Simulation

- Spectra
- Flux
- Power Density
- 3D Power Density (including CAD)
- Multiple beams
- Time dependent  $E(t)$  and  $B(t)$  fields



# About OSCARS

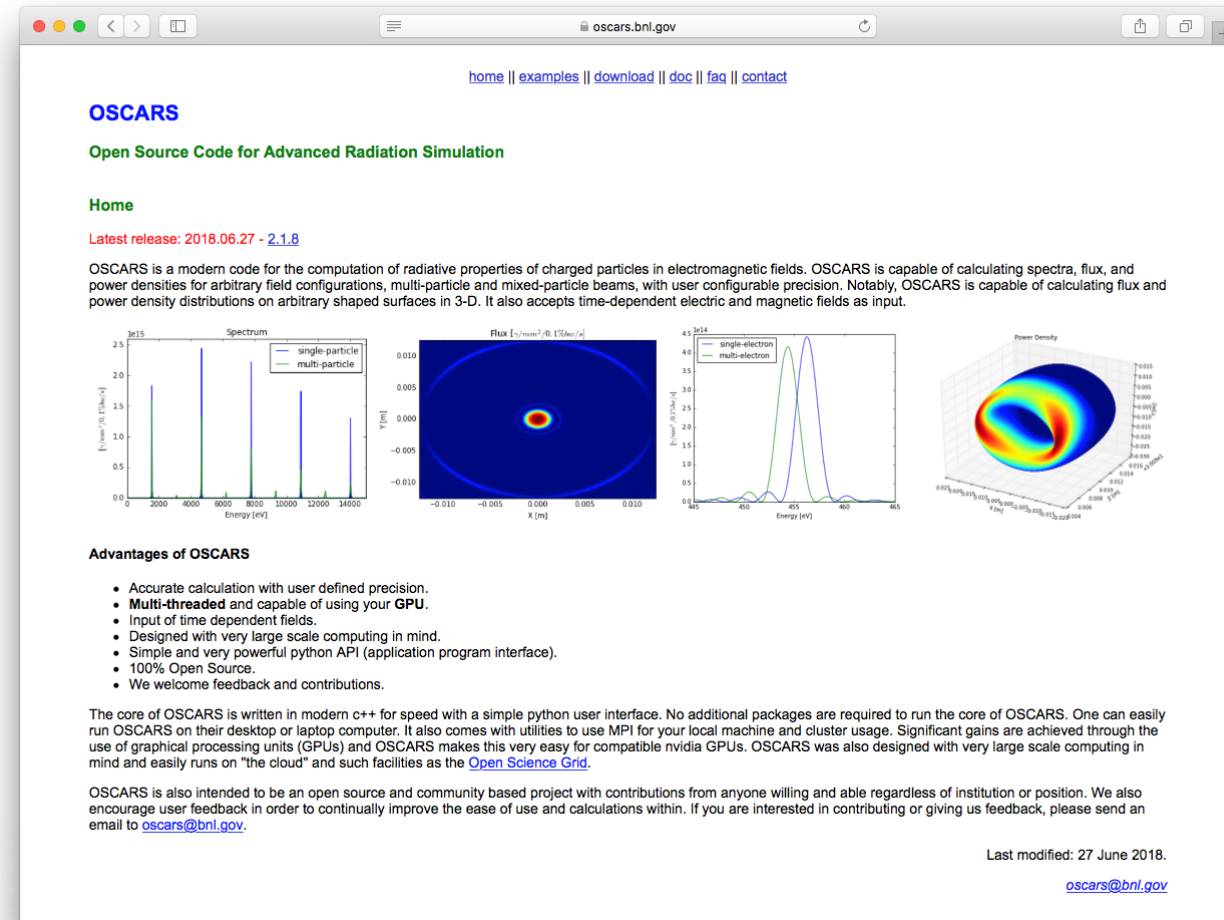
- Multi-threaded internally
  - Simply add argument: `nthreads=72` for faster calculation
- Capable of using Multiple-GPUs simultaneously
  - Simply add: `gpu=1` (for all available), or `gpus=[0, 1, 4]` to specify which GPUs to use
  - Using CUDA, GPUs communicate directly (direct GPU-GPU data transfers)
- MPI compatible, Used on large-scale grid computing (*i.e.* Open Science Grid)
- Written in c++ (for speed) with python API
  - Technically: C-Python extension



# Where to find OSCARS

- Anywhere (pypi)
  - **pip install oscars**
- Anywhere (conda)
  - **conda install oscars -c lightsource2-tag**
- <https://oscars.bnl.gov>
- [oscars@bnl.gov](mailto:oscars@bnl.gov)
- <https://github.com/dhidas/OSCARS>
- Full documentation and many examples

- Linux ✓
- OS X ✓
- Windows ✓

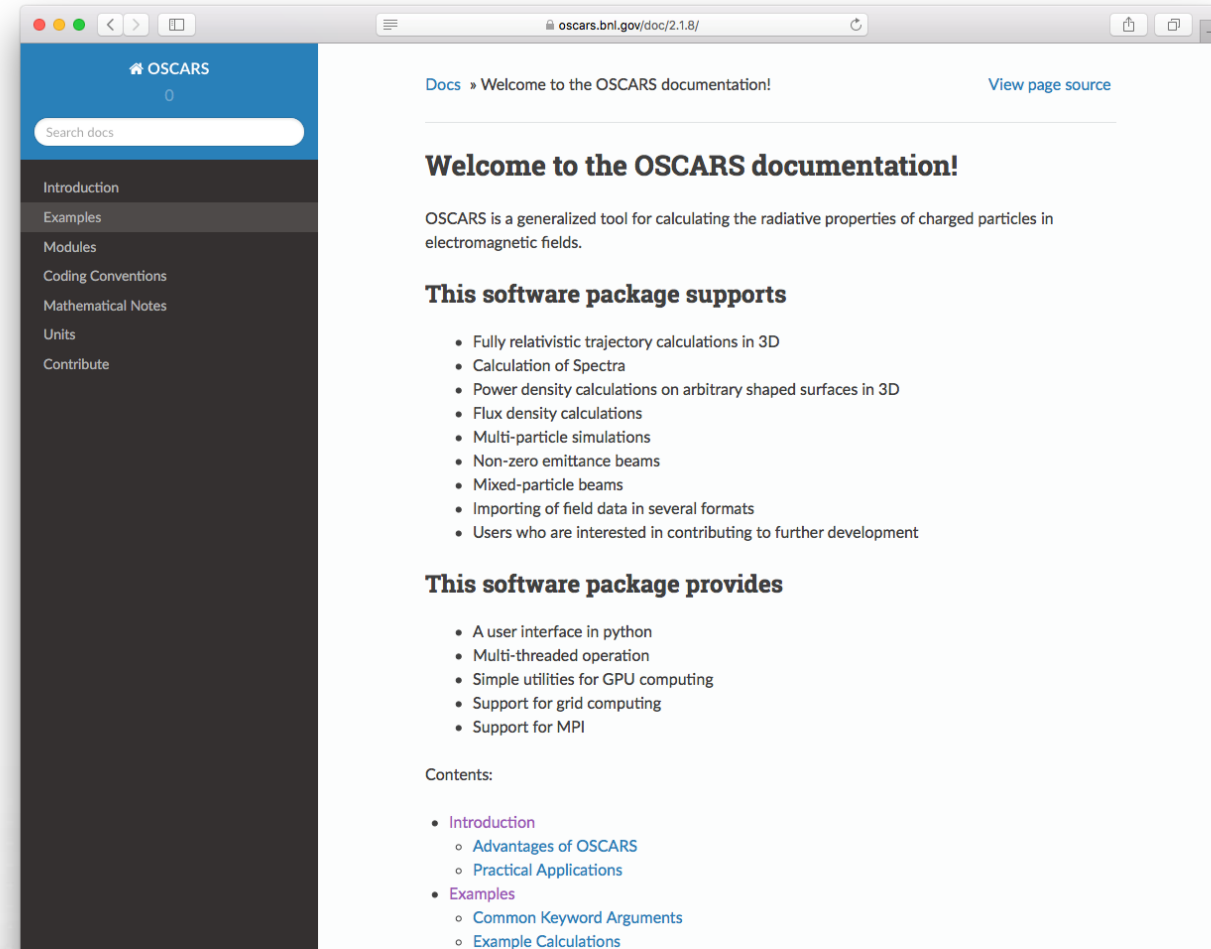


The screenshot shows the OSCARS website homepage. At the top, there are navigation links: [home](#) || [examples](#) || [download](#) || [doc](#) || [faq](#) || [contact](#). The main heading is "OSCARS" followed by "Open Source Code for Advanced Radiation Simulation". Below this, it says "Home" and "Latest release: 2018.06.27 - 2.1.8". The main text describes OSCARS as a modern code for computing radiative properties of charged particles in electromagnetic fields. It lists capabilities like calculating spectra, flux, and power densities for arbitrary field configurations, multi-particle and mixed-particle beams, and 3-D power density distributions. There are four plots: a bar chart titled "Spectrum" showing energy vs. flux for single and multi-particle cases; a 2D heatmap titled "Flux" showing flux density in the X-Y plane; a line graph titled "Spectrum" showing flux vs. energy for single and multi-electron cases; and a 3D surface plot titled "Power Density" showing power density in 3D space. Below the plots, there is a section "Advantages of OSCARS" with a bulleted list: Accurate calculation with user defined precision, Multi-threaded and capable of using your GPU, Input of time dependent fields, Designed with very large scale computing in mind, Simple and very powerful python API (application program interface), 100% Open Source, and We welcome feedback and contributions. At the bottom, there is a paragraph about the core of OSCARS being written in modern c++ and a paragraph about its open source nature and community-based project. The footer includes "Last modified: 27 June 2018." and the email [oscars@bnl.gov](mailto:oscars@bnl.gov).

# Where to find OSCARS

- Anywhere (pypi)
  - **pip install oscar**s
- Anywhere (conda)
  - **conda install oscar**s **-c lightsources2-tag**
- <https://oscar.s.bnl.gov>
- [oscar.s@bnl.gov](mailto:oscar.s@bnl.gov)
- <https://github.com/dhidas/OSCAR>
- Full **documentation** and many examples

- Linux ✓
- OS X ✓
- Windows ✓



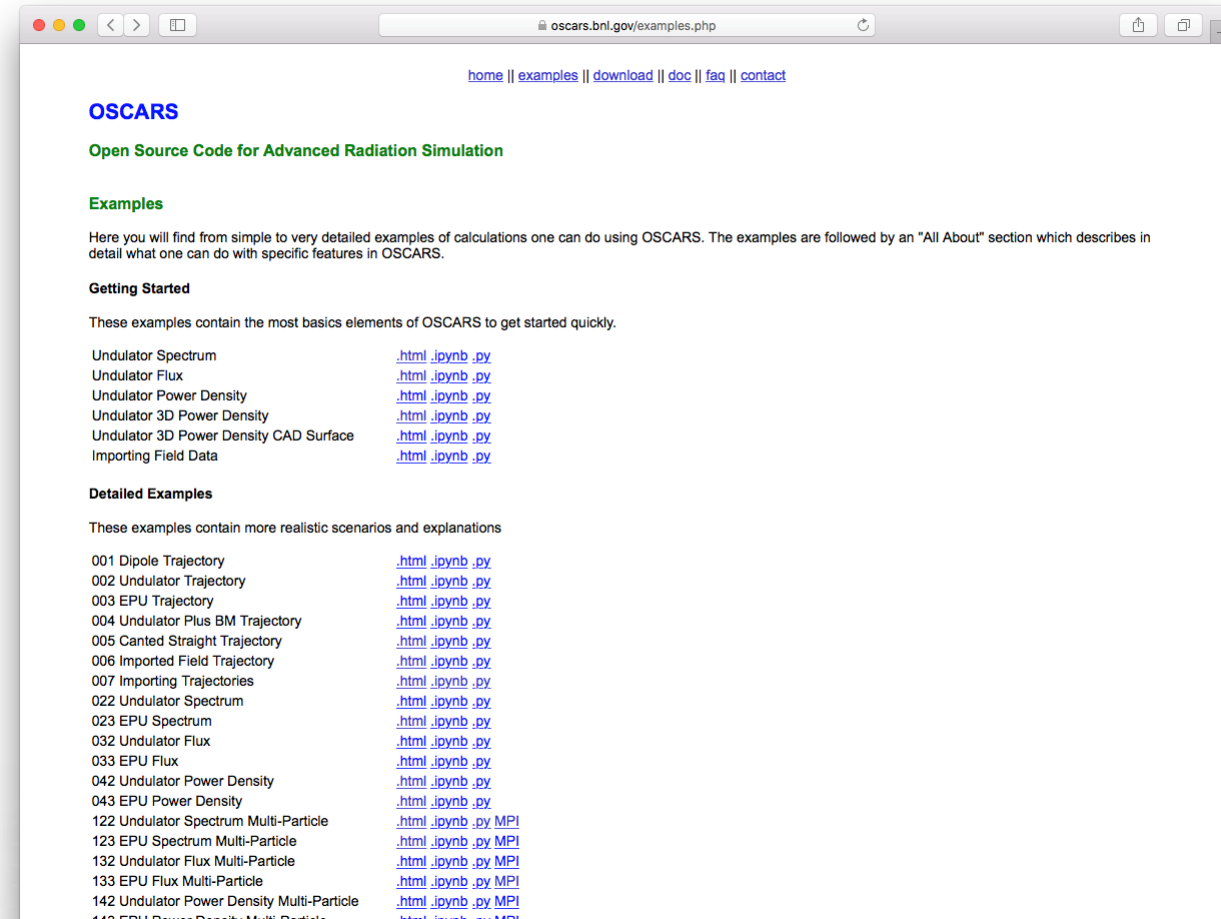
The screenshot shows a web browser window displaying the OSCARS documentation page. The browser's address bar shows the URL `oscar.s.bnl.gov/doc/2.1.8/`. The page has a dark blue header with the OSCARS logo and a search bar. A sidebar on the left contains a navigation menu with items: Introduction, Examples, Modules, Coding Conventions, Mathematical Notes, Units, and Contribute. The main content area is white and features a heading "Welcome to the OSCARS documentation!" followed by a brief description of the tool. Below this, there are two sections: "This software package supports" and "This software package provides", each with a bulleted list of features and capabilities. At the bottom, there is a "Contents:" section with a list of links to various parts of the documentation.



# Where to find OSCARS

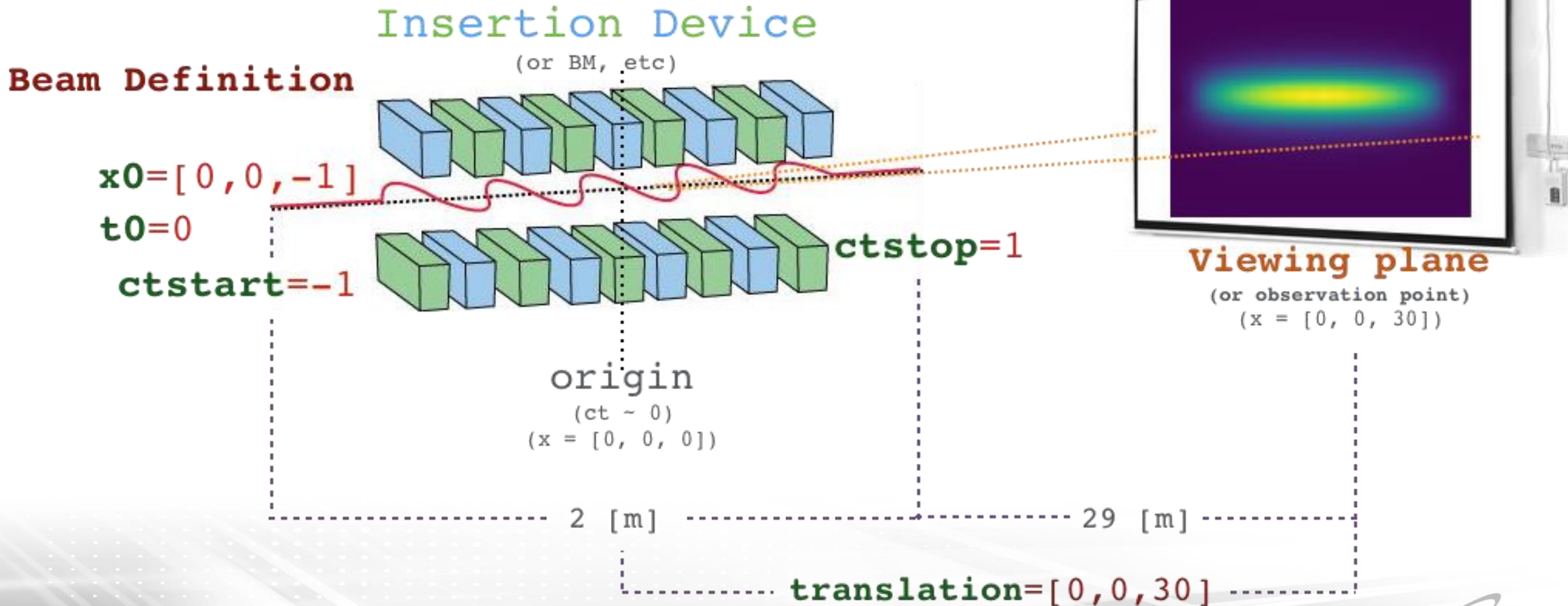
- Anywhere (pypi)
  - **pip install oscars**
- Anywhere (conda)
  - **conda install oscars -c lightsource2-tag**
- <https://oscars.bnl.gov>
- [oscars@bnl.gov](mailto:oscars@bnl.gov)
- <https://github.com/dhidas/OSCARS>
- Full documentation and many **examples**

- Linux ✓
- OS X ✓
- Windows ✓



# SR Simulation Construction

- Elements of an SR Simulation



# SR Full Simulation (from previous page)

Should be easy, here to help you

```
import oscars.sr
osr = oscars.sr.sr(gpu=1,
                  nthreads=16)

osr.set_particle_beam(type='electron',
                      energy_GeV=3,
                      ctstartstop=[-1, 1])

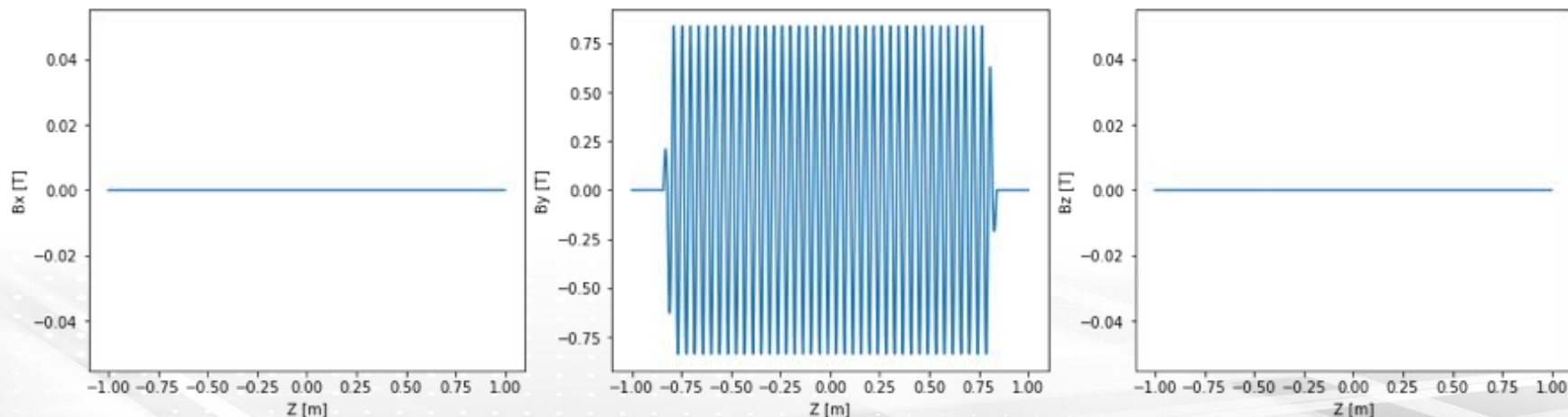
osr.add_bfield_undulator(bfield=[0, 0.8375, 0],
                          period=[0, 0, 0.042],
                          nperiods=38)

osr.calculate_power_density_rectangle(plane='XY',
                                      width=[0.04, 0.04],
                                      npoints=[101, 101],
                                      translation=[0, 0, 30])
```

# Magnetic Fields

- Many types of built-in fields
  - Undulator, bending magnet, gaussian, quadrupole, arbitrary Python function, data files, interpolated data
- Add a simulated undulator

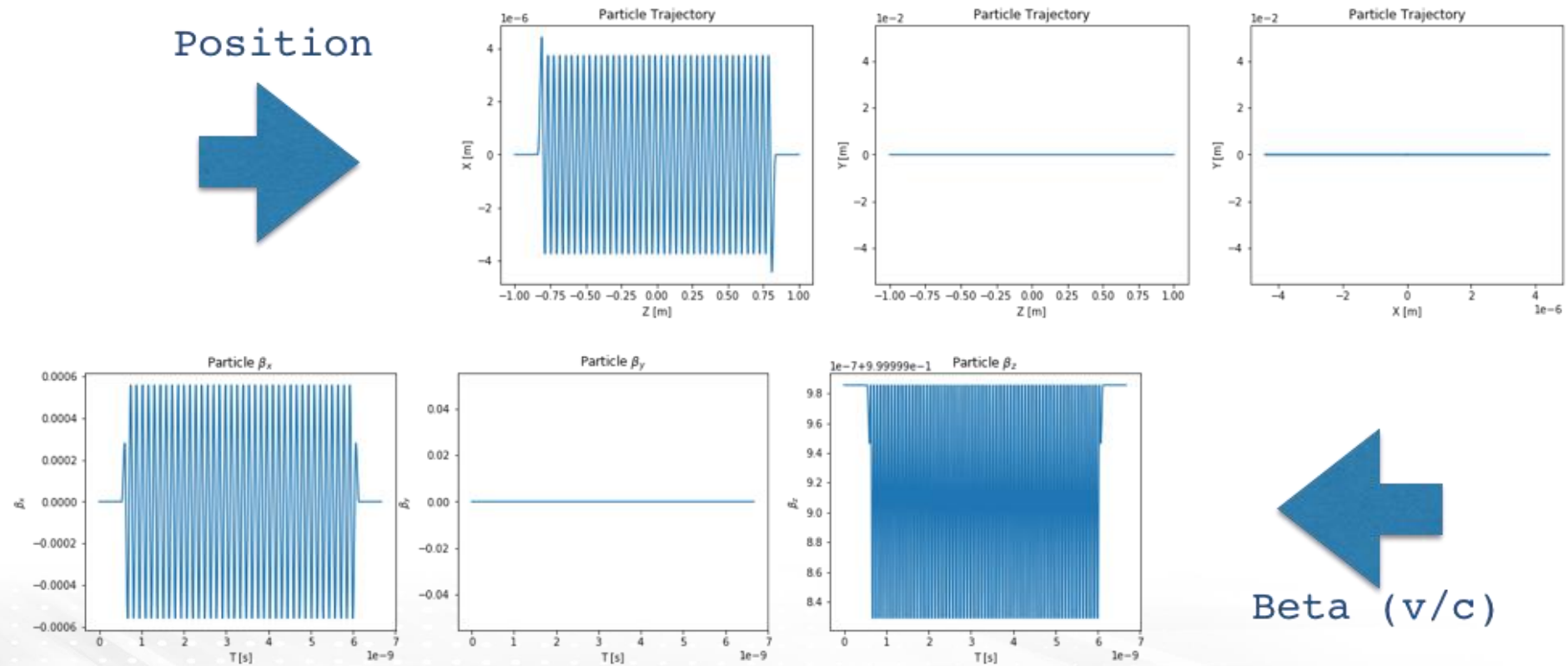
```
osr.add_bfield_undulator(bfield=[0, 0.8375, 0], period=[0, 0, 0.042], nperiods=38)
```



# Calculate Trajectory

- Calculate trajectory as a sanity check (not necessary)

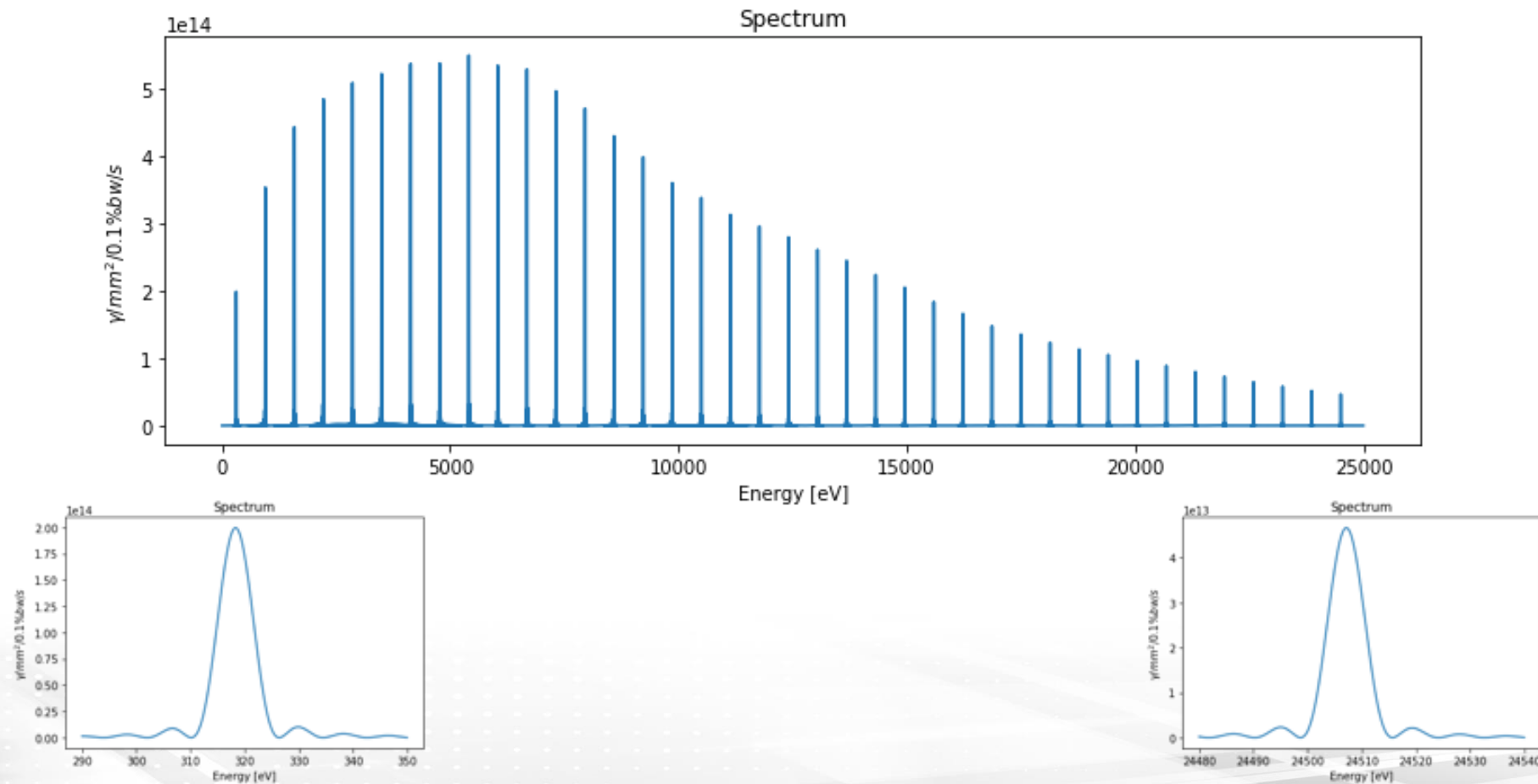
`osr.calculate_trajectory()`



# Calculating Spectra

- Calculate spectrum at observation point

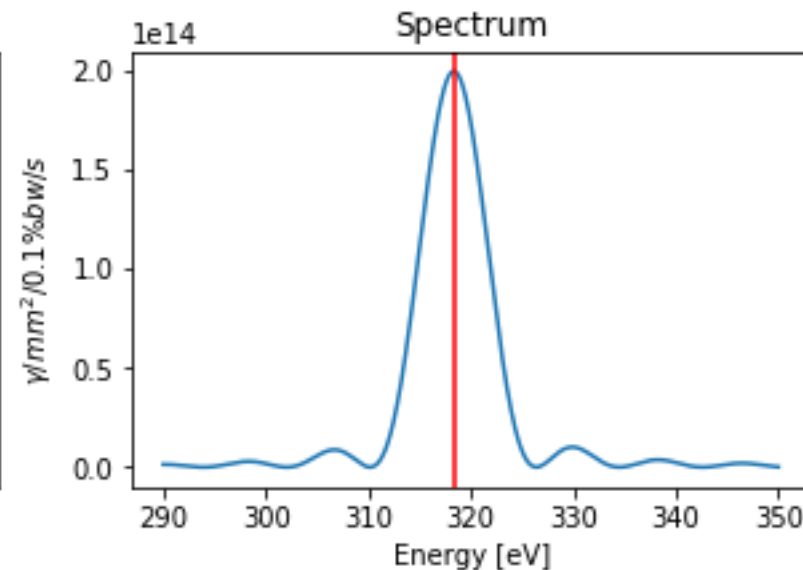
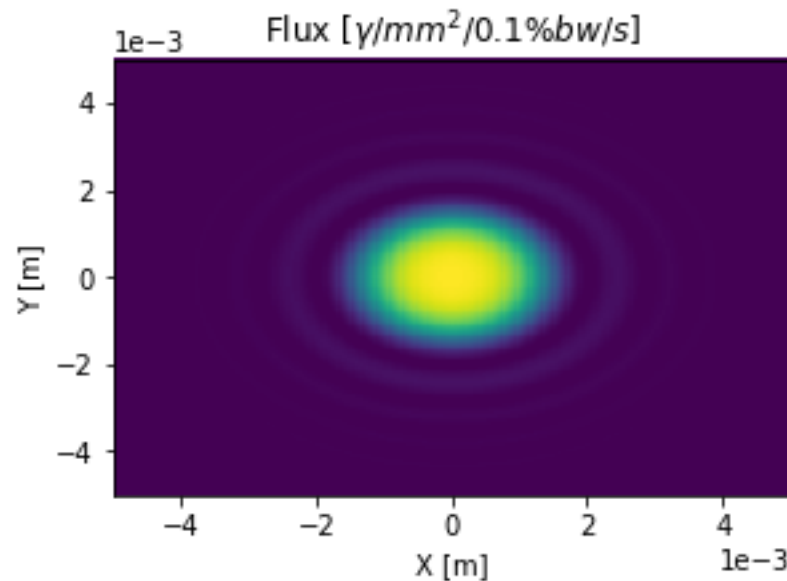
```
osr.calculate_spectrum(obs=[0, 0, 30], energy_range_eV=[10, 25000])
```



# Calculating Flux

- Calculating the flux at specific wavelength on an observation plane

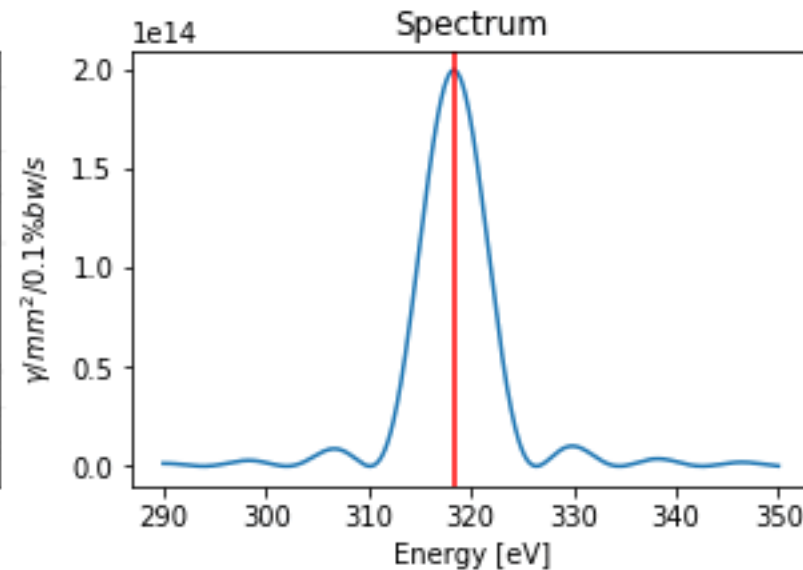
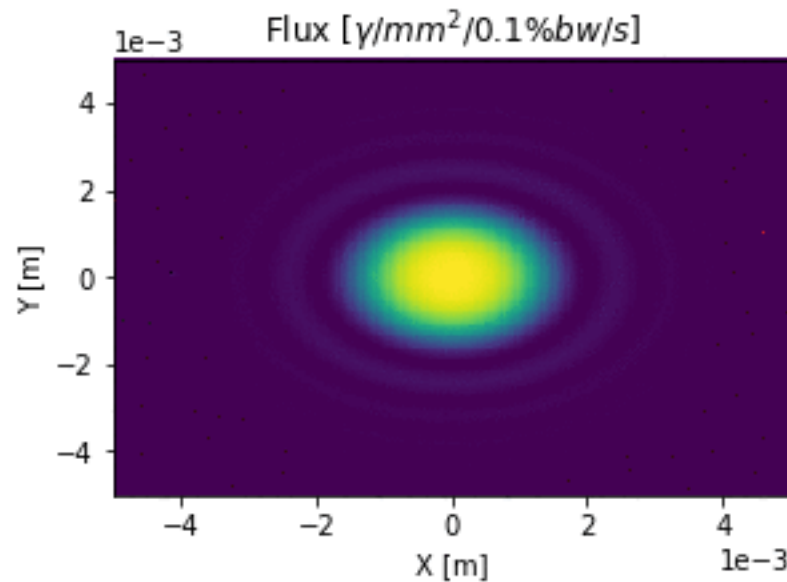
```
osr.calculate_flux_rectangle(plane='XY',  
                             energy_eV=318,  
                             width=[0.01, 0.01],  
                             npoints=[101, 101],  
                             translation=[0, 0, 30])
```



# Calculating Flux

- Calculating the flux at specific wavelength on an observation plane

```
osr.calculate_flux_rectangle(plane='XY',  
                             energy_eV=318,  
                             width=[0.01, 0.01],  
                             npoints=[101, 101],  
                             translation=[0, 0, 30])
```

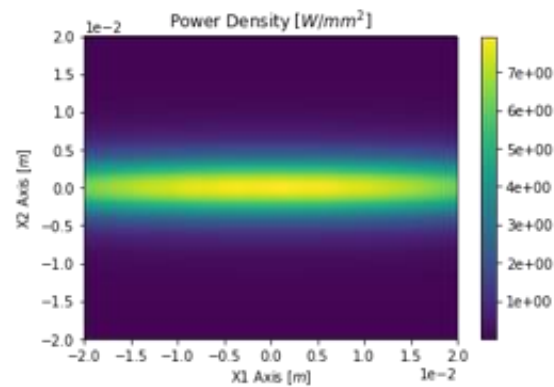
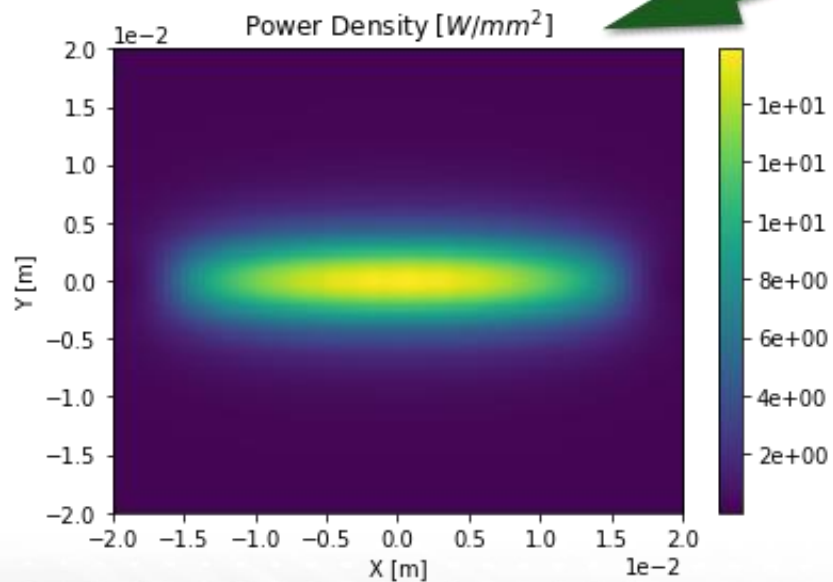




# Calculating Power Density

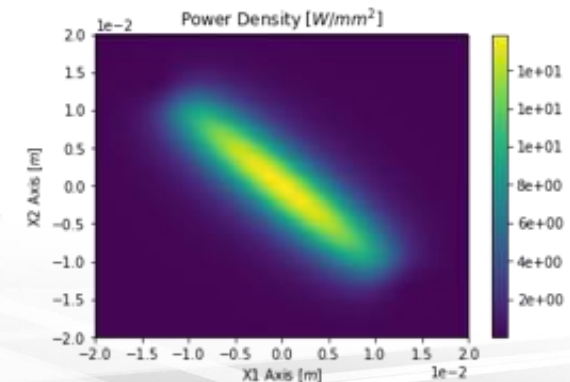
- Calculate power density on observation plane

```
osr.calculate_power_density_rectangle(plane='XY',  
width=[0.04, 0.04],  
npoints=[101, 101],  
translation=[0, 0, 30])
```



```
rotations=[0, osr.pi()/3, 0]
```

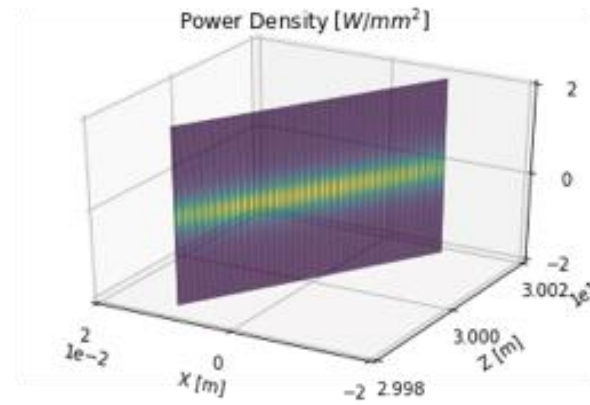
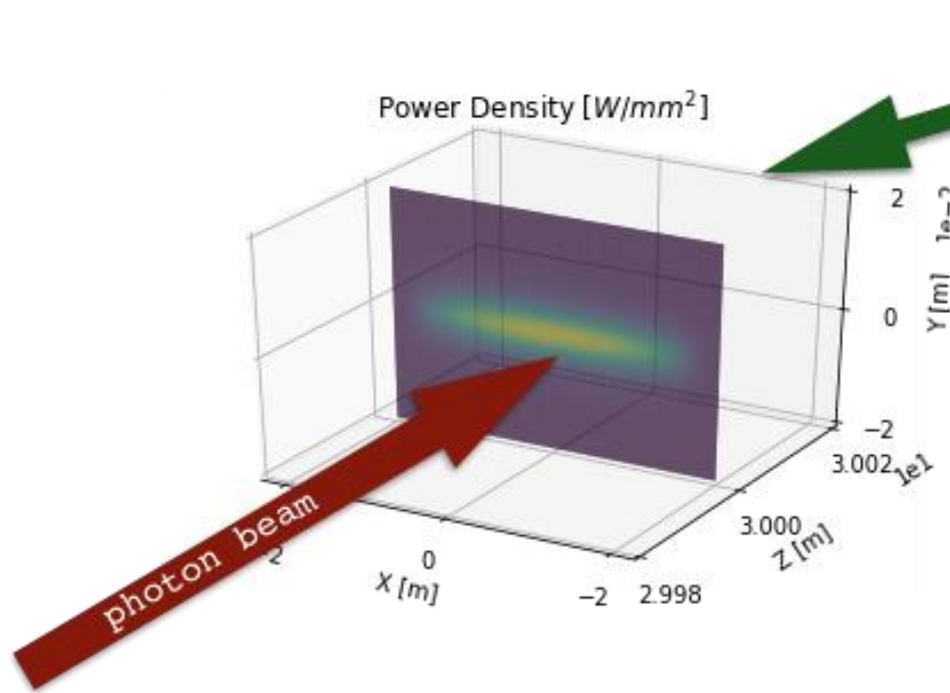
```
rotations=[0, 0, osr.pi()/4]
```



# Calculating Power Density

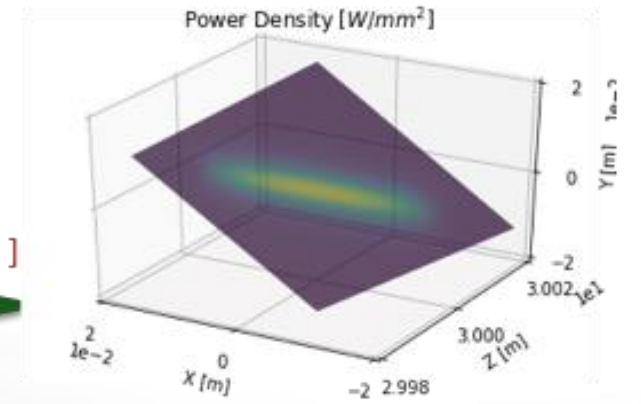
- Calculate power density on observation plane (3D)

```
osr.calculate_power_density_rectangle(plane='XY',  
width=[0.04, 0.04],  
npoints=[101, 101],  
translation=[0, 0, 30])
```



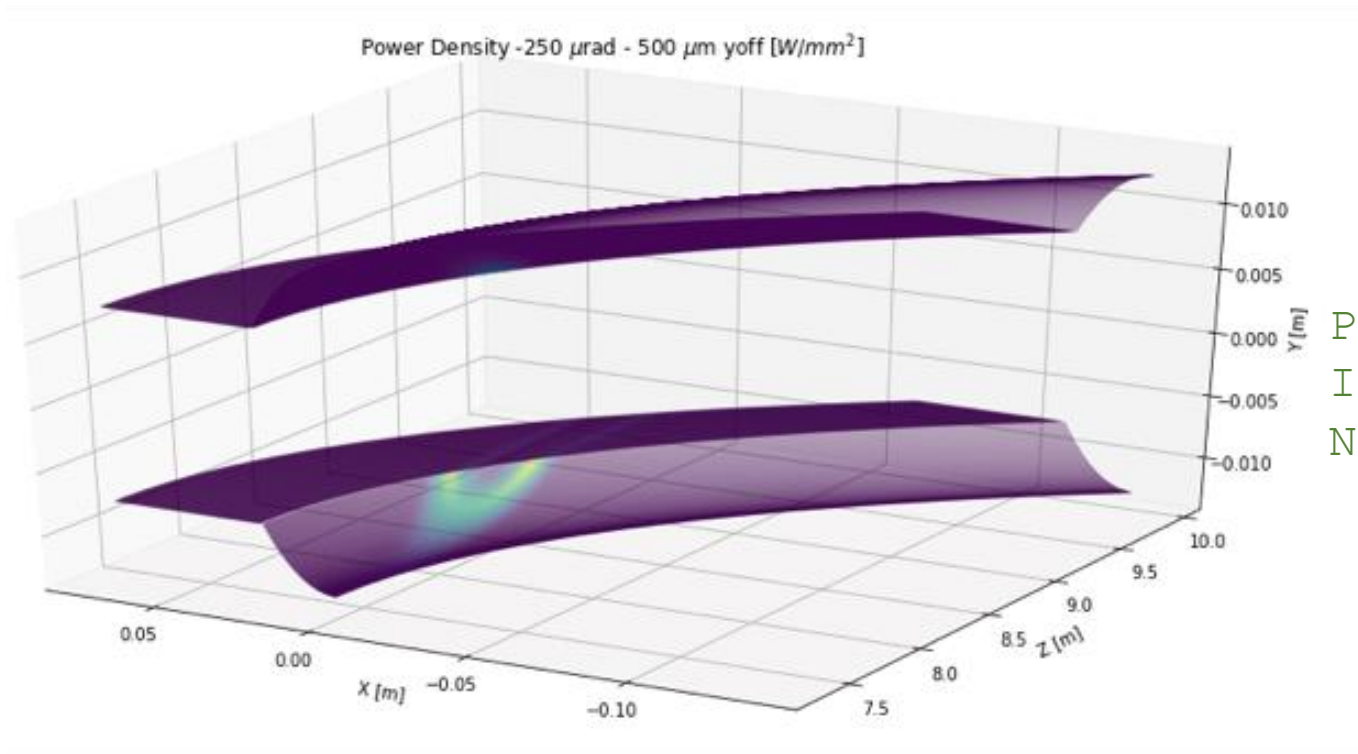
```
rotations=[0, osr.pi()/3, 0]
```

```
rotations=[0, 0, osr.pi()/4]
```



# Power Density – Parametric 3D

- Advantage: Much easier visualization

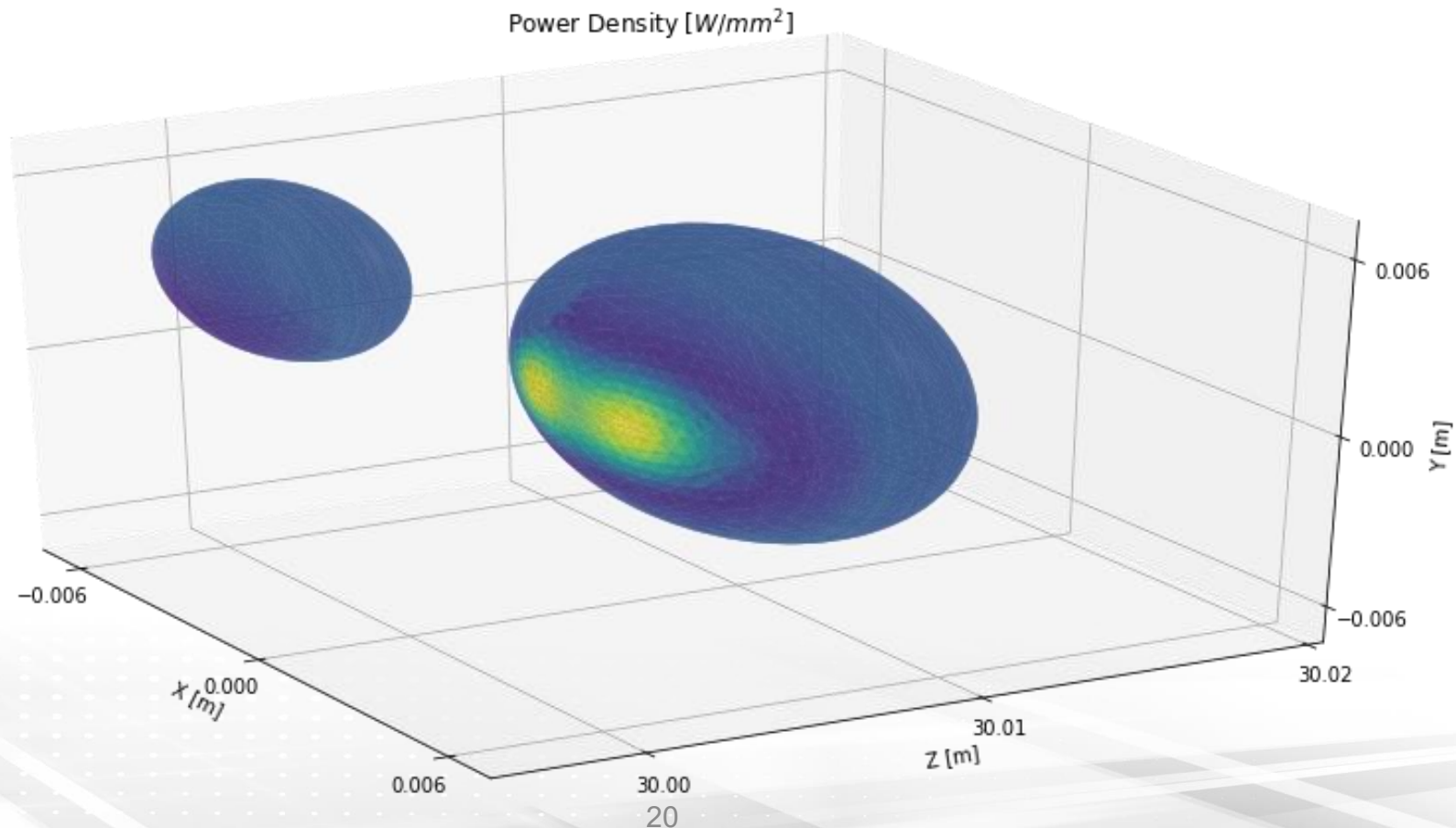


Power Density  
Inner wall,  
NSLS-II Dipole chamber

- Major disadvantage: Very difficult for complex objects

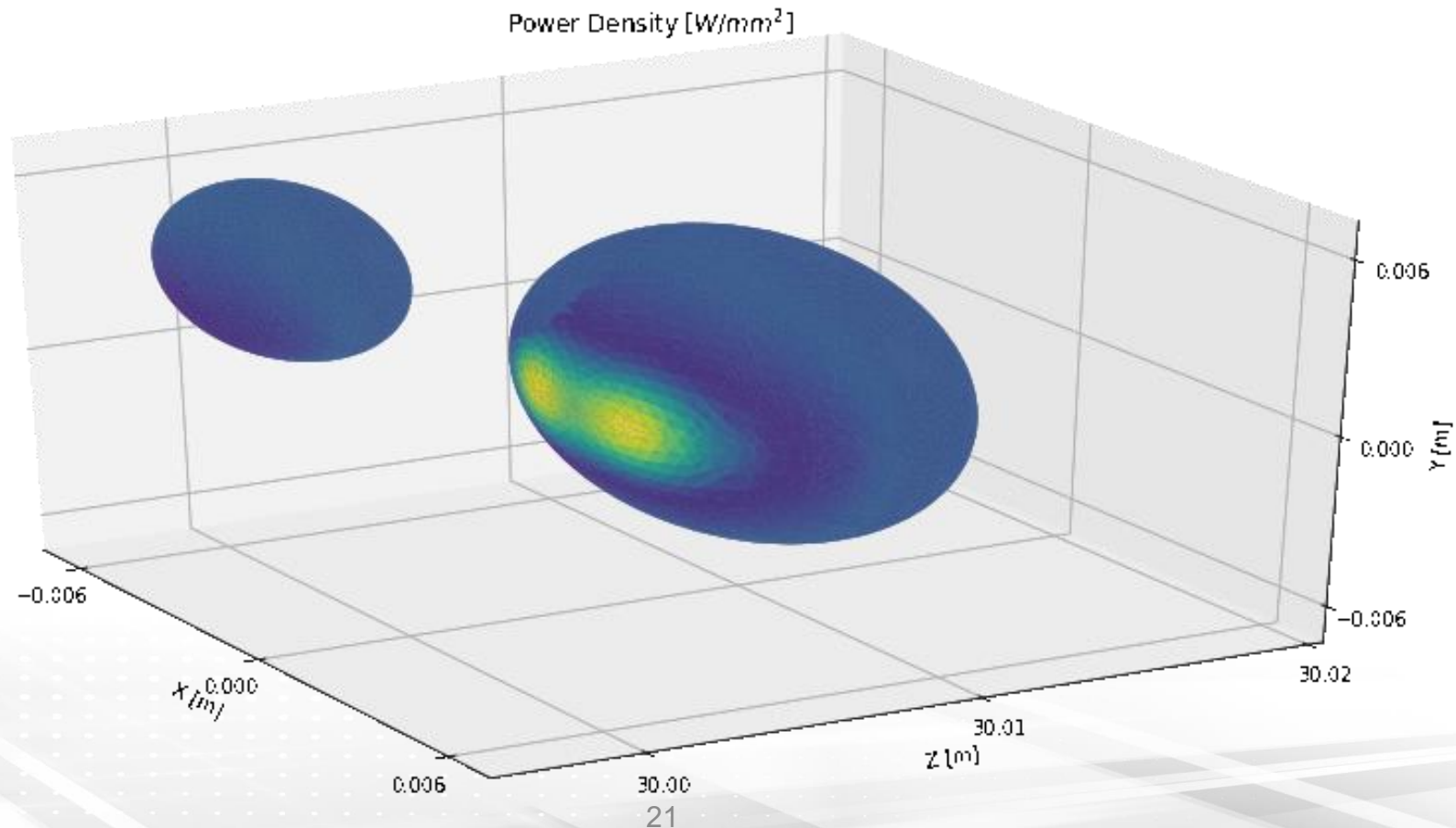
# Power Density CAD

- Can now import CAD model in STL format
- Allows for substantially more complex objects (than shown here)



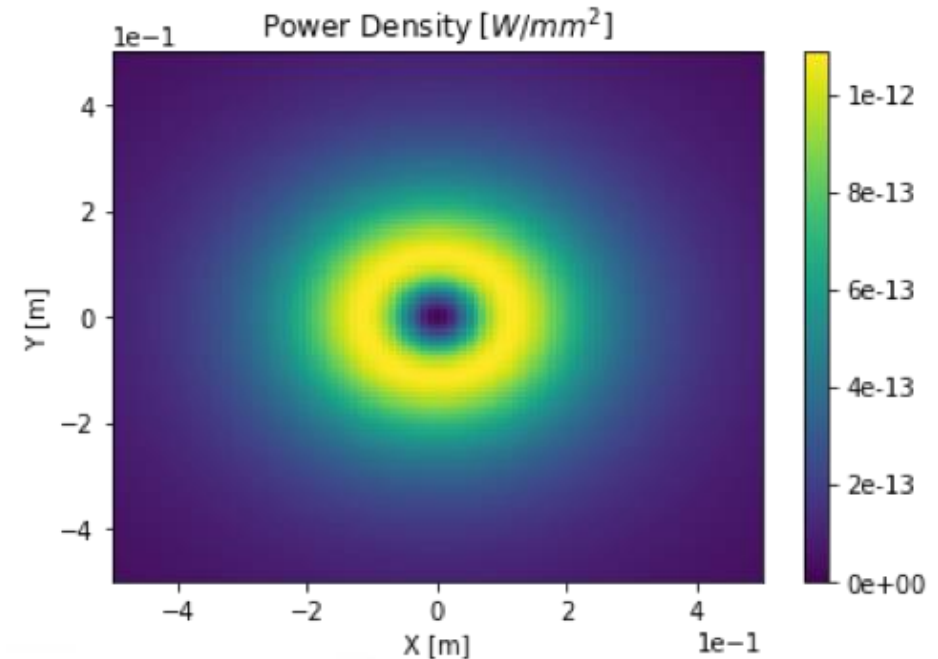
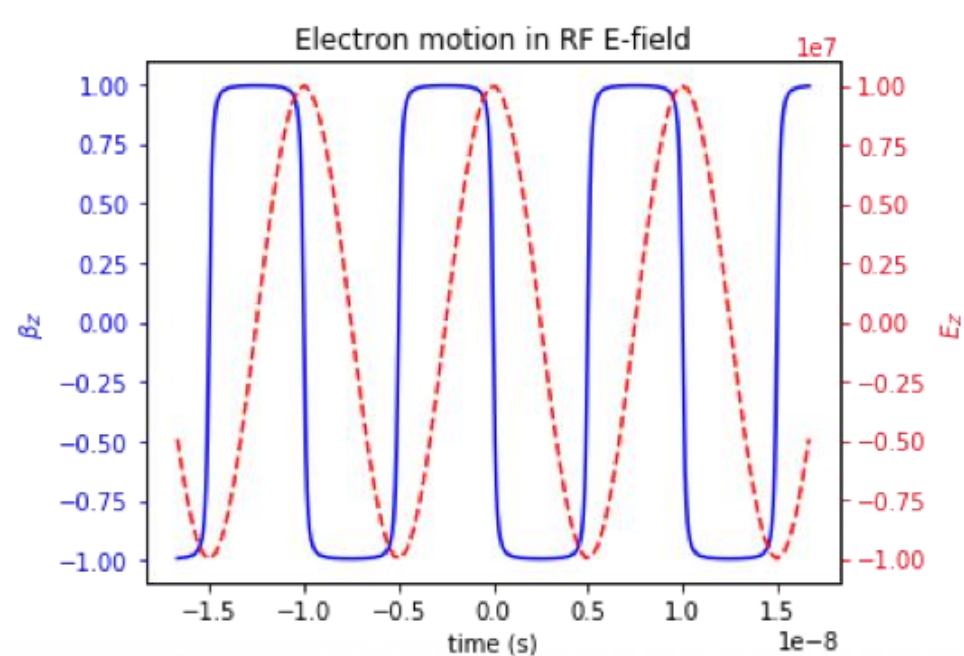
# Power Density CAD

- Can now import CAD model in STL format
- Allows for substantially more complex objects (than shown here)



# Time Dependence

- Any functional form  $B(x, y, z, t)$ ,  $E(x, y, z, t)$  or real field data in resonance
- E.g. 100 [MHz], 10 [MV/m]



- Also valid for very high fields  $> 100$  [GV/m]

# Conclusion

- Synchrotron Radiation Basics
- Introduction to OSCARS and basic calculations
- Calculations on 3D Geometries
- Time Dependent E, B
- Visit, contribute: <https://oscars/bnl.gov>

**Thank You**