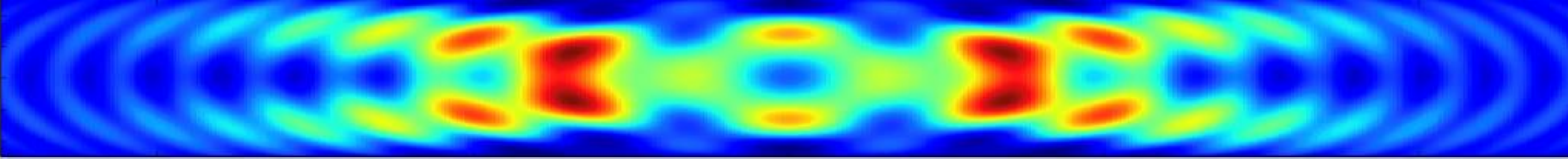


# Novel, Fast, Open-Source Code for Synchrotron Radiation Computation on Arbitrary 3D Geometries

Dean Andrew Hidas

BROOKHAVEN NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

# Outline

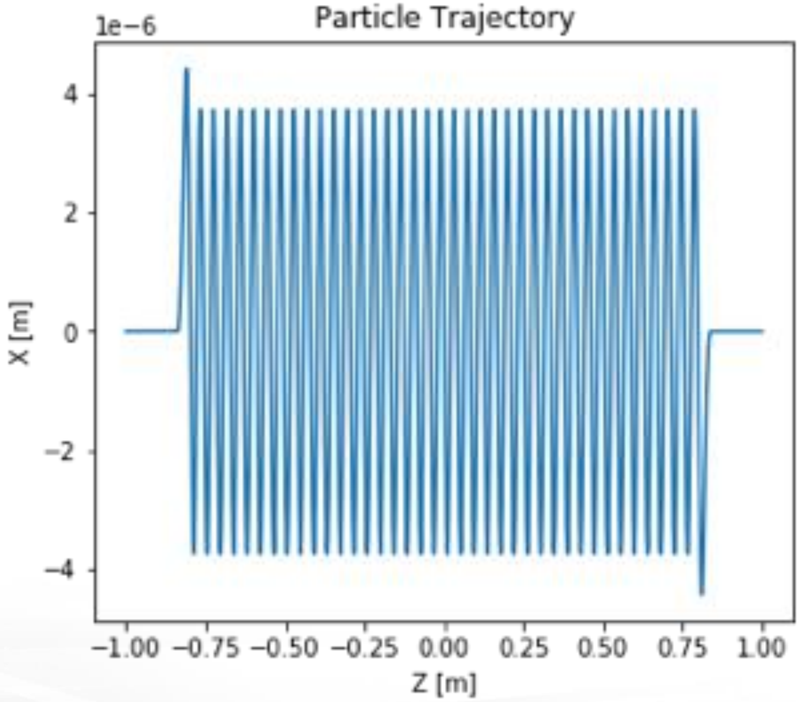
- Synchrotron Radiation Basics
- Introduction to OSCARS
- Basic SR Calculations
- Calculations on 3D Geometries
- Time Dependent E, B

# 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
  - $2^{nd}$ order differential equation

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

    - Typically solved via $4^{th}$ order Runge-Kutta method



# SR Basics

- Calculate a quantity of interest (from trajectory information)

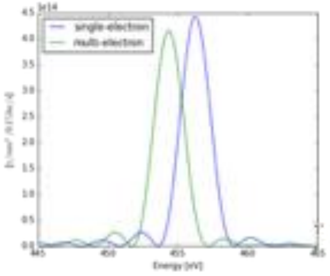
- Flux, from E-field

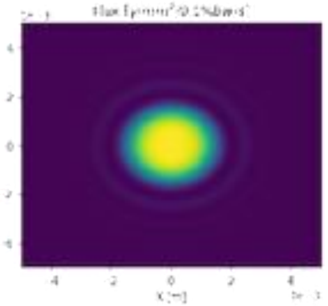
$$\vec{E}(\vec{x},\omega) = C\int_{-\infty}^{+\infty}\left[\frac{1}{\gamma^2}\frac{\hat{n}-\vec{\beta}}{R^2(1-\hat{n}\cdot\vec{\beta})^2} + \frac{\hat{n}\times(\hat{n}-\vec{\beta})\times\dot{\vec{\beta}}}{R(1-\hat{n}\cdot\vec{\beta})^2}\right]e^{i\omega(\tau+R/c)}\mathrm{d}\tau$$

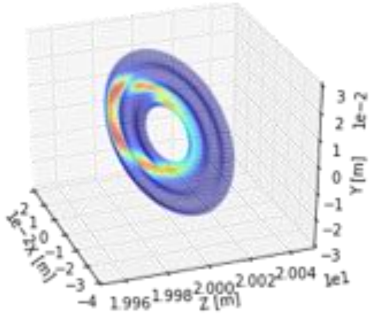
- Numerically integrate this (and the like) to desired accuracy
    - Smaller trajectory steps = higher accuracy, cost is time and memory
    - GPU and multi-threading good candidates for this

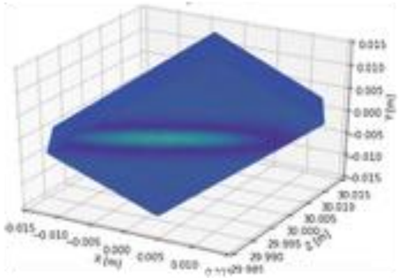
# Introduction to OSCARS

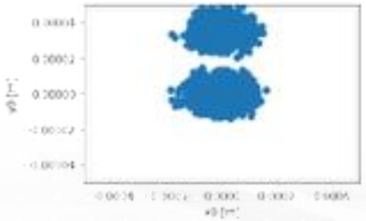
- **O**pen **S**ource **C**ode for **A**dvanced **R**adiation **S**imulation
  - Spectra
  - Flux
  - Power Density
  - 3D Power Density (including CAD)
  - Multiple beams
  - Time dependent E(t) and B(t) fields

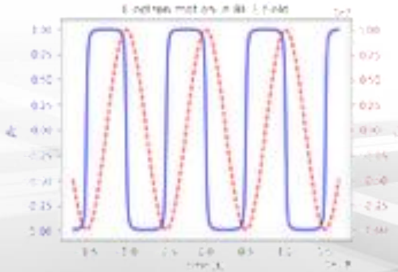












# About OSCARS

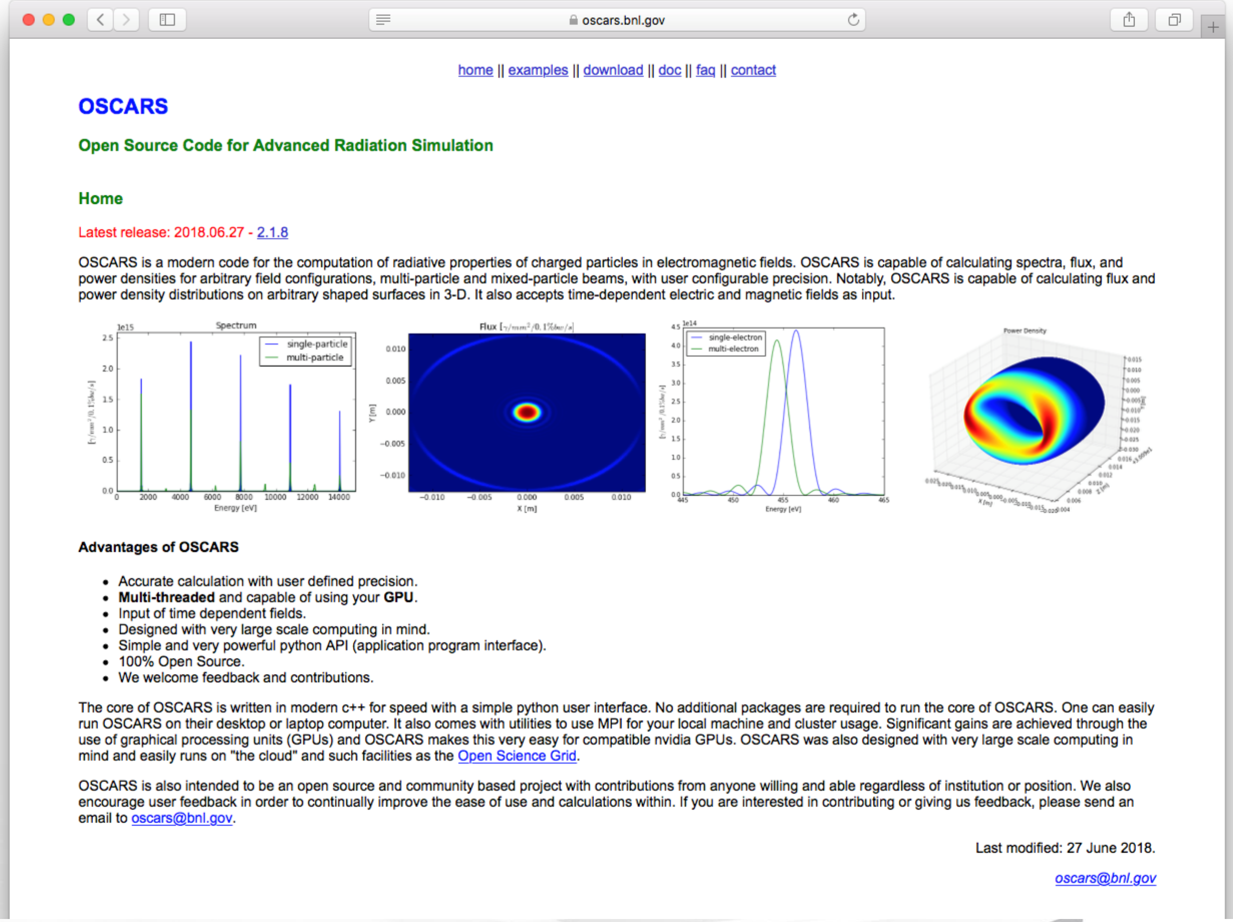
- Multi-threaded internally
  - Simply add argument: nthreads=72 for faster calculation


- 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

- Linux ✓
- OS X ✓
- Windows ✓

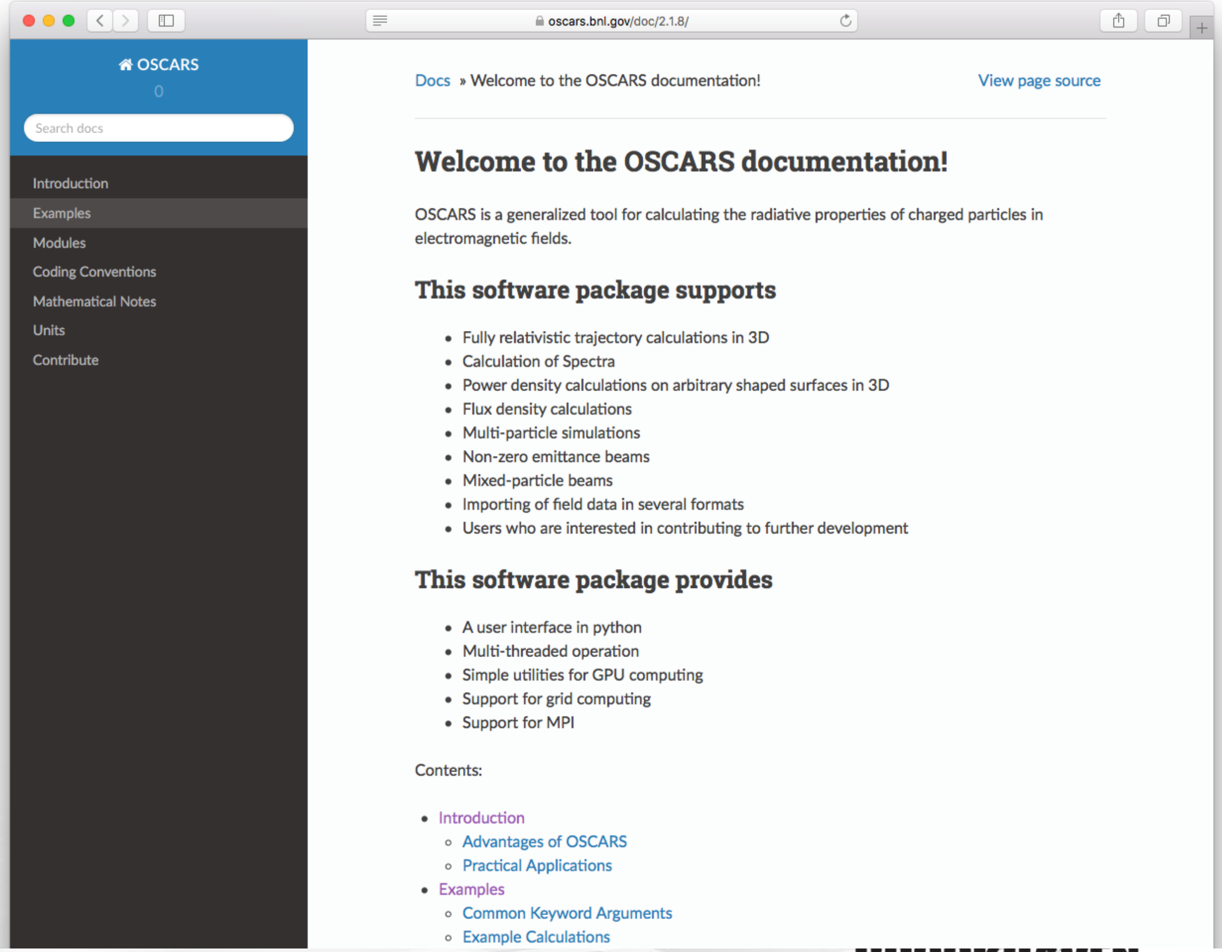
- Anywhere (pypi)
  - **pip install oscars**
- Anywhere (conda)
  - **conda install oscars –c lightsource2-tag**
- https://oscars.bnl.gov
- oscars@bnl.gov
- https://github.com/dhidas/OSCARS
- Full documentation and many examples



# Where to find OSCARS

- Linux
- OS X
- Windows

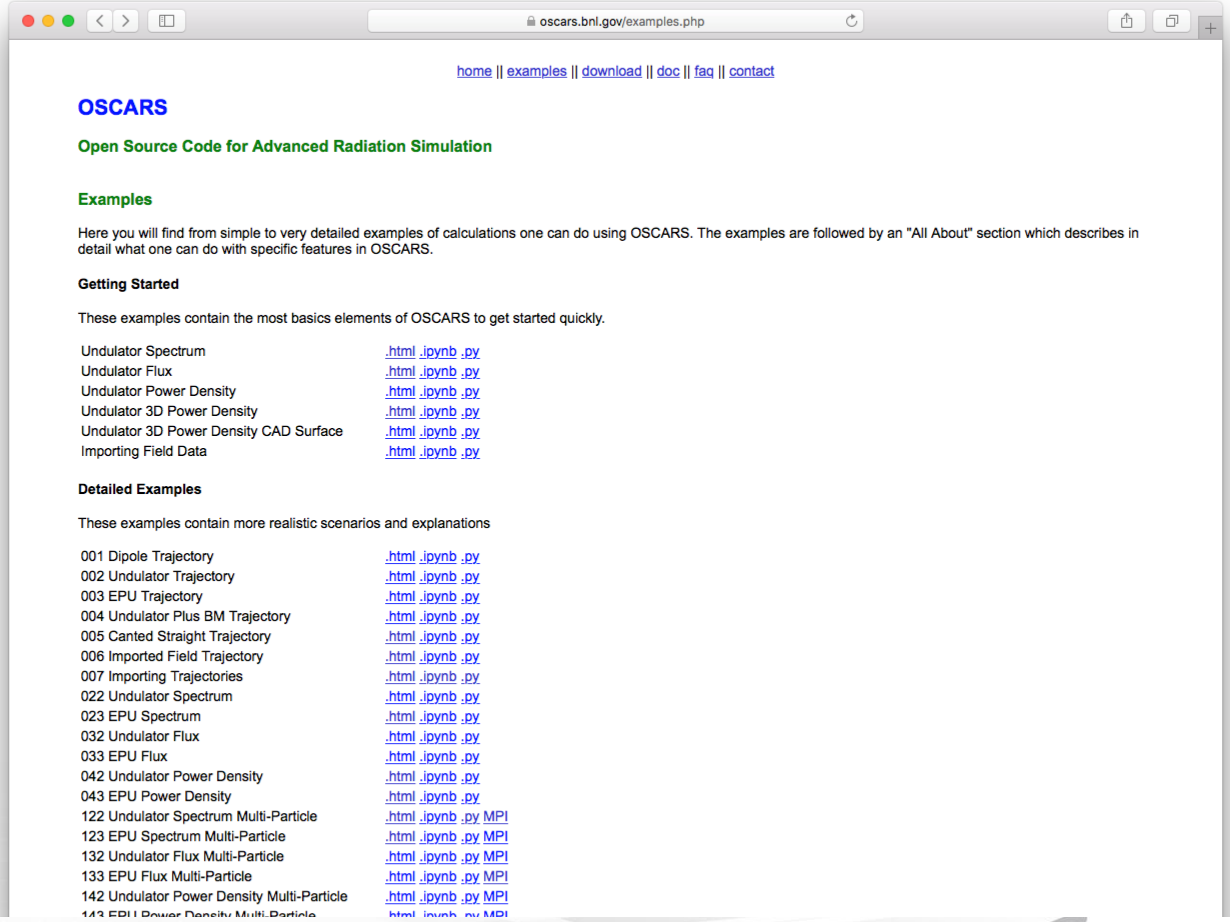
- Anywhere (pypi)
  - **pip install oscars**
- Anywhere (conda)
  - **conda install oscars –c lightsource2-tag**
- https://oscars.bnl.gov
- oscars@bnl.gov
- https://github.com/dhidas/OSCARS
- Full **documentation** and many examples



# Where to find OSCARS

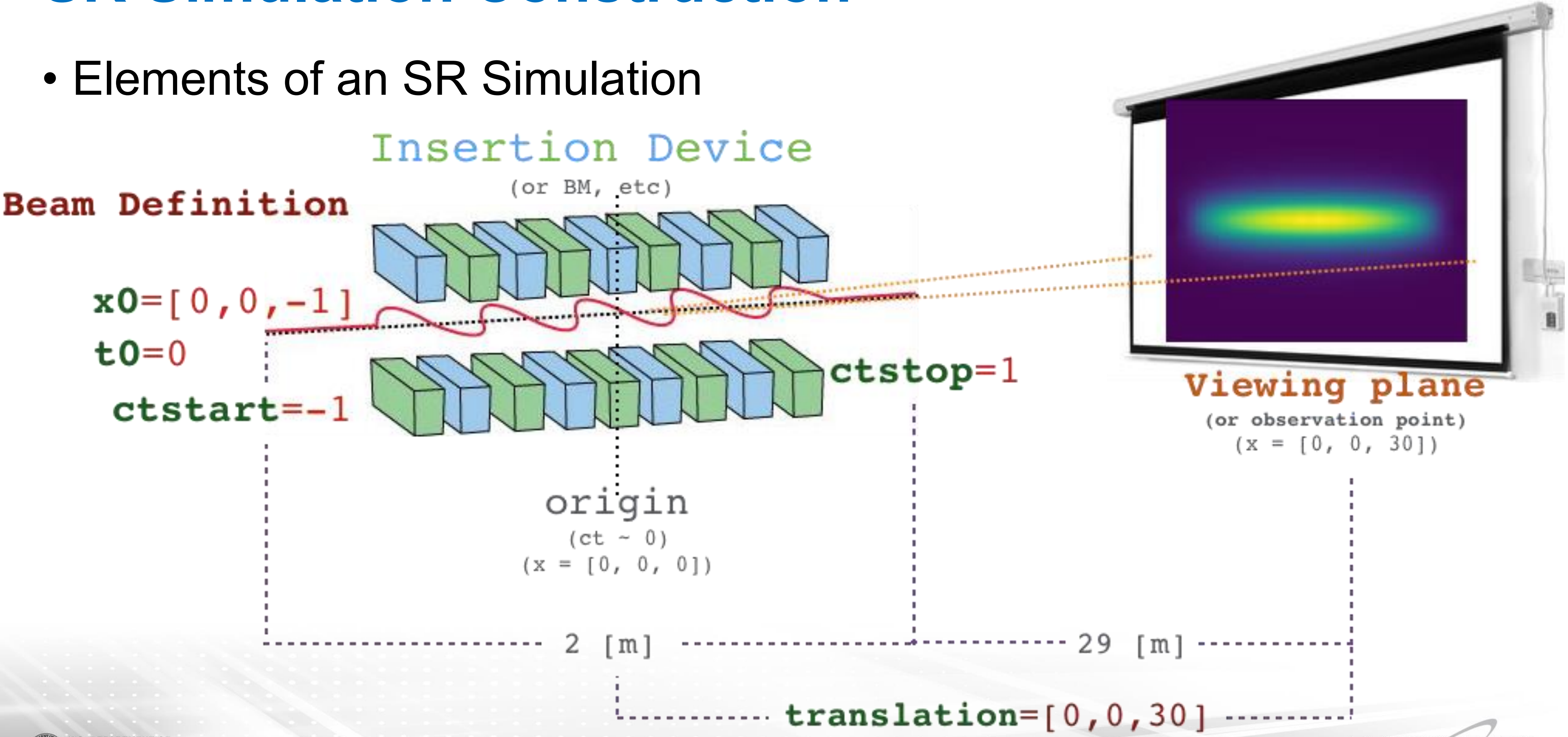
- Linux
- OS X
- Windows

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



# SR Simulation Construction

- Elements of an SR Simulation



# SR Full Simulation (from previous page)

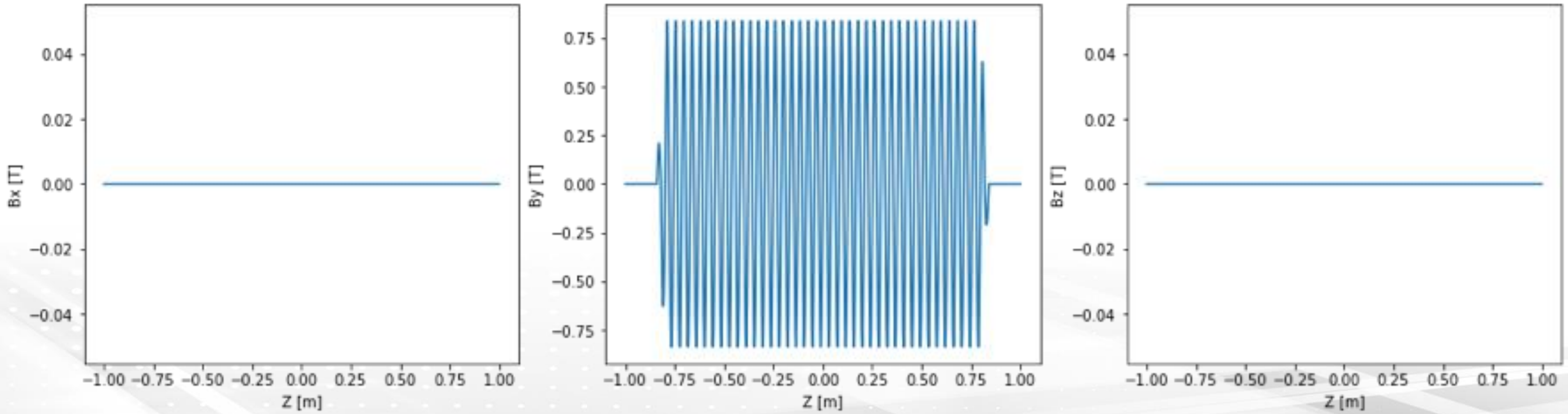
Should be easy, here to help you

```
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)
```





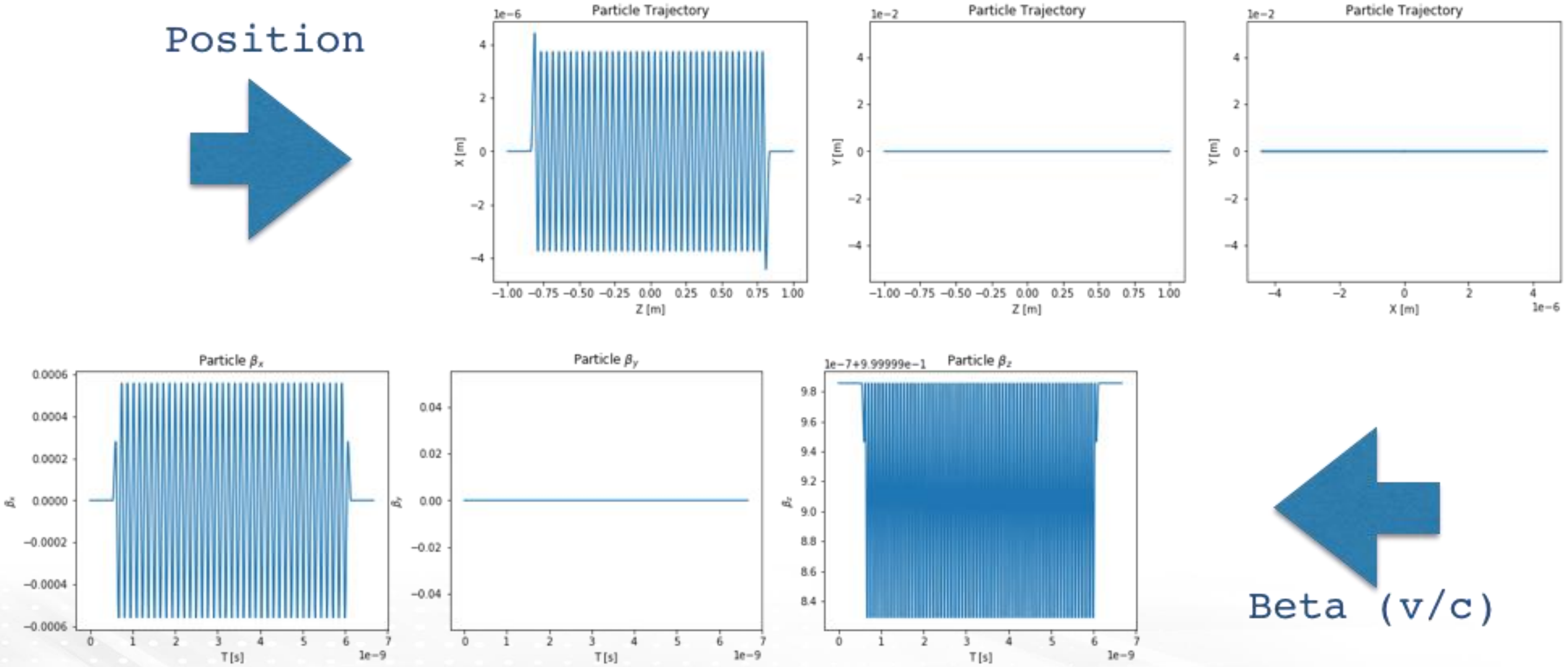

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY

# Calculate Trajectory

- Calculate trajectory as a sanity check (not necessary)

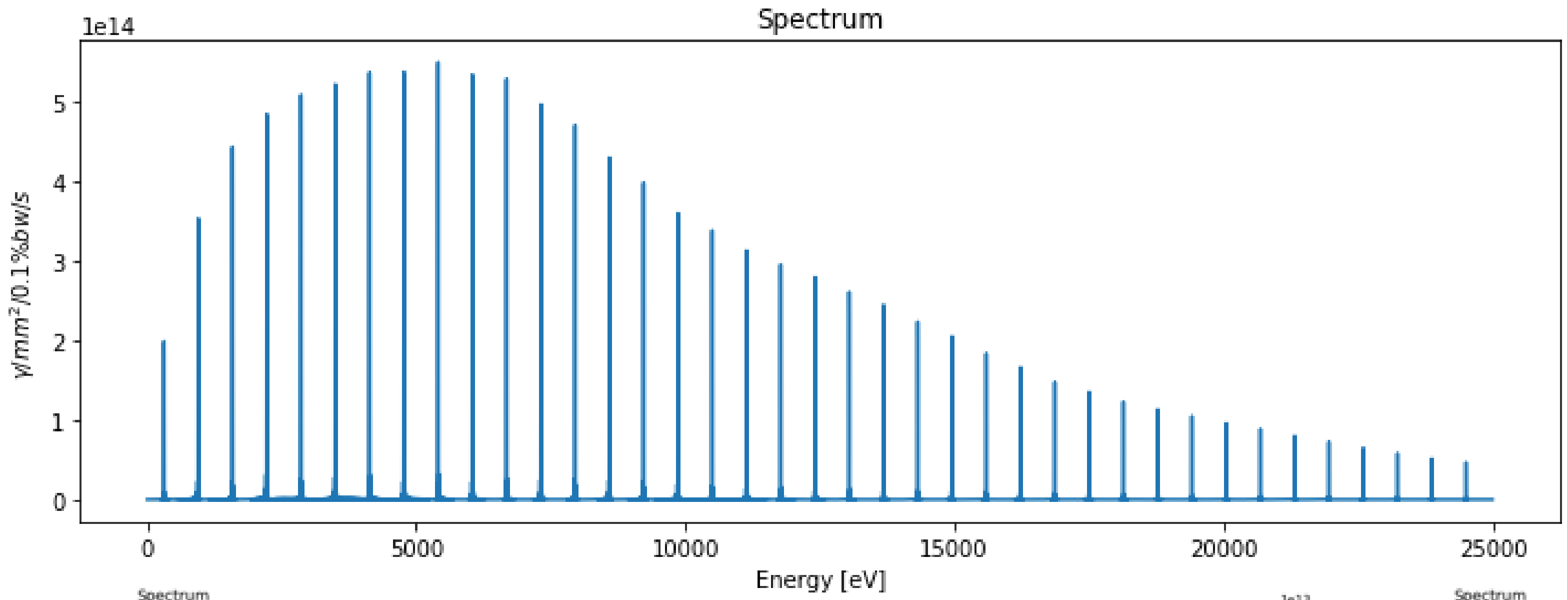
```
osr.calculate_trajectory()
```

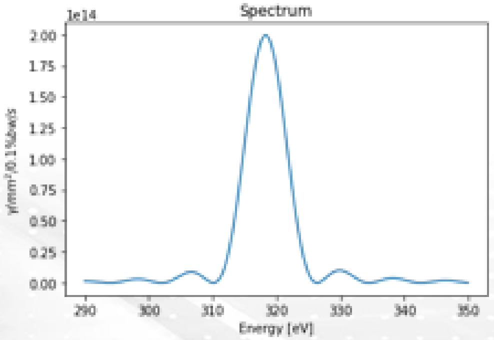


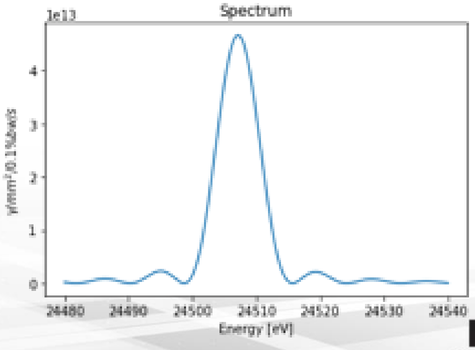
# Calculating Spectra

- Calculate spectrum at observation point

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



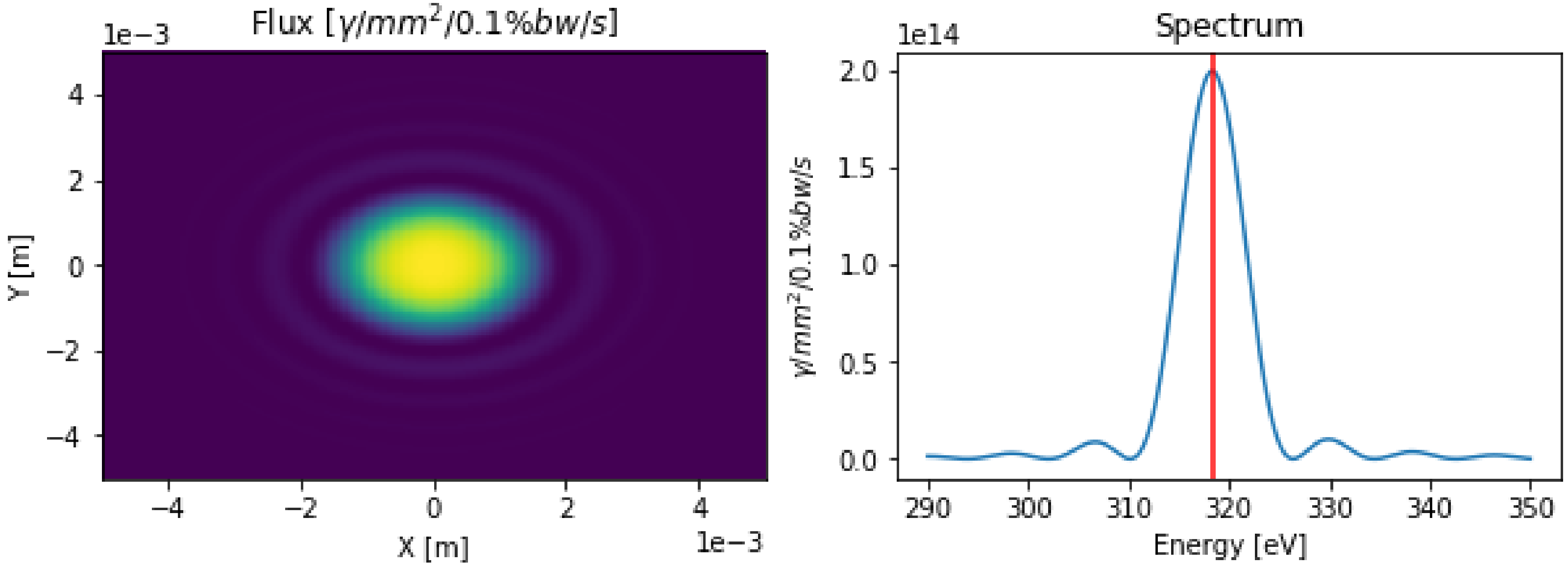



# Calculating Flux

- Calculating the flux at specific wavelength on an observation plane

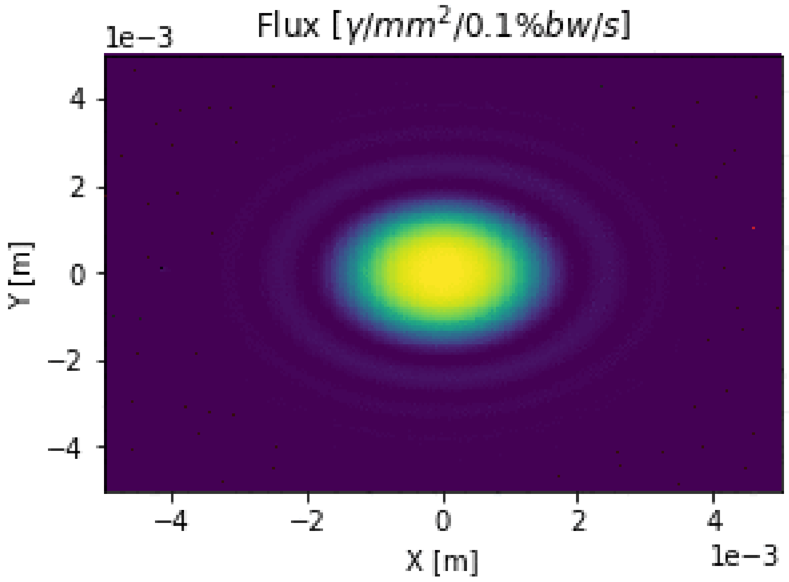
```
osr.calculate_flux_rectangle(plane='XY',
                             energy_eV=318,
                             width=[0.01, 0.01],
                             npoints=[101, 101],
                             translation=[0, 0, 30])
```

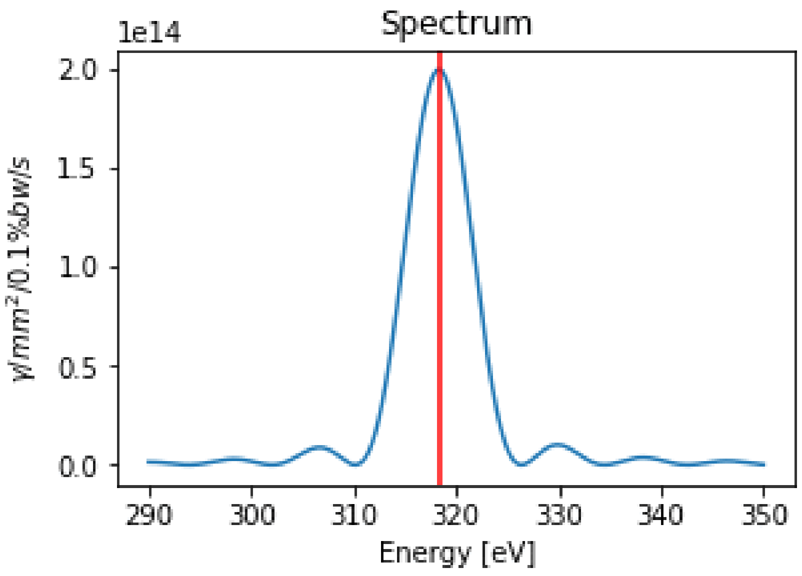


# Calculating Flux

- 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])
```

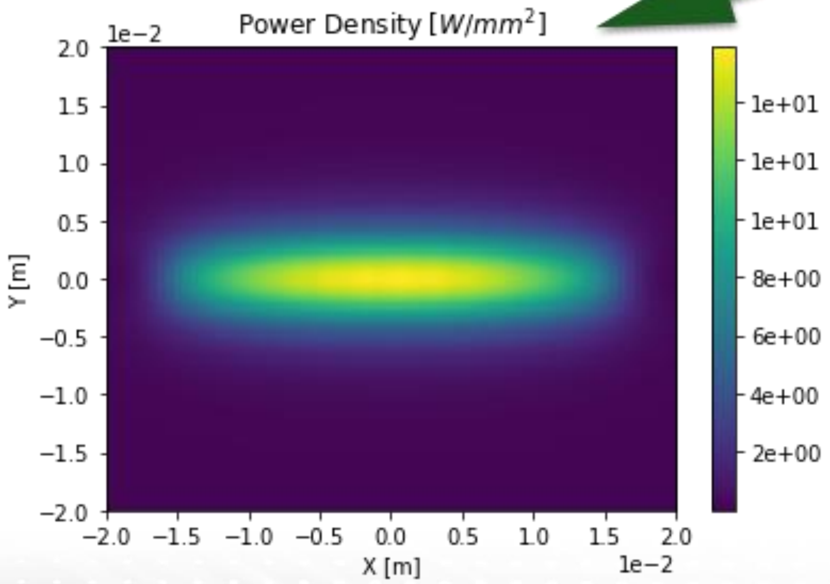


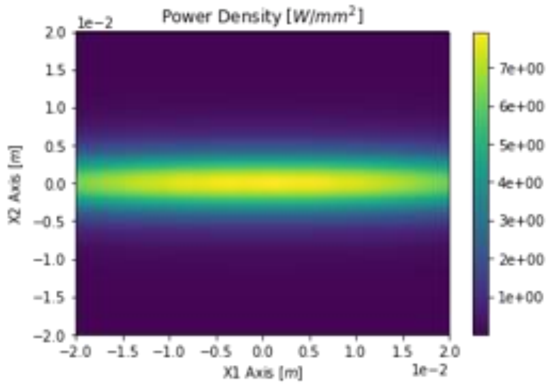



# Calculating Power Density

- Calculate power density on observation plane

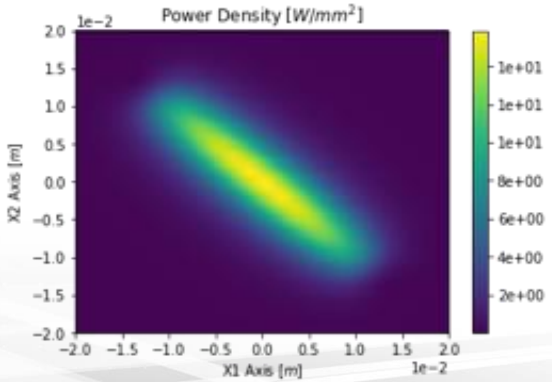
```
osr.calculate_power_density_rectangle(plane='XY',
                                      width=[0.04, 0.04],
                                      npoints=[101, 101],
                                      translation=[0, 0, 30])
```




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

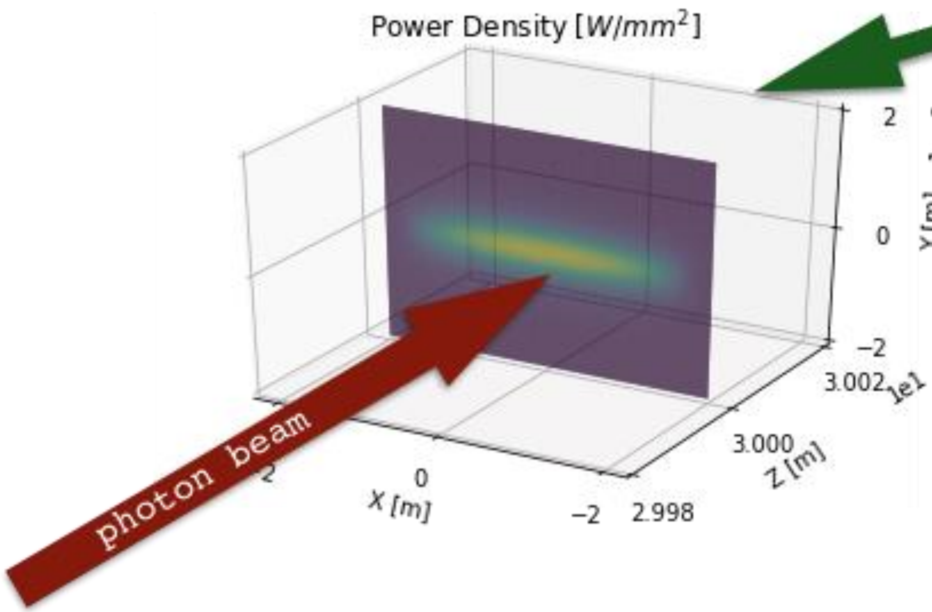
`rotations=[0, 0, osr.pi()/4]`

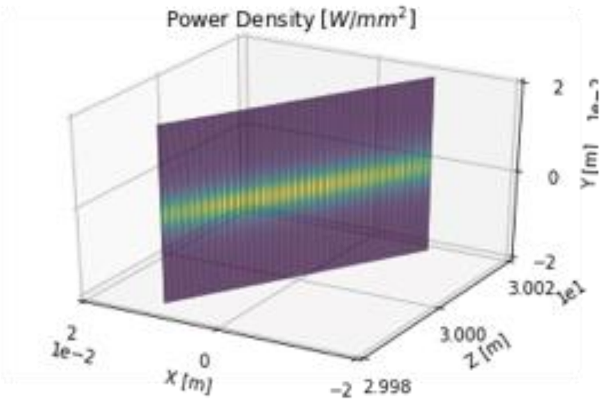


# Calculating Power Density

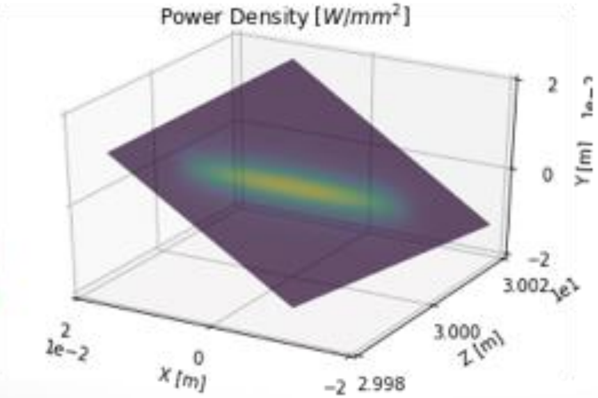
- Calculate power density on observation plane (3D)

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



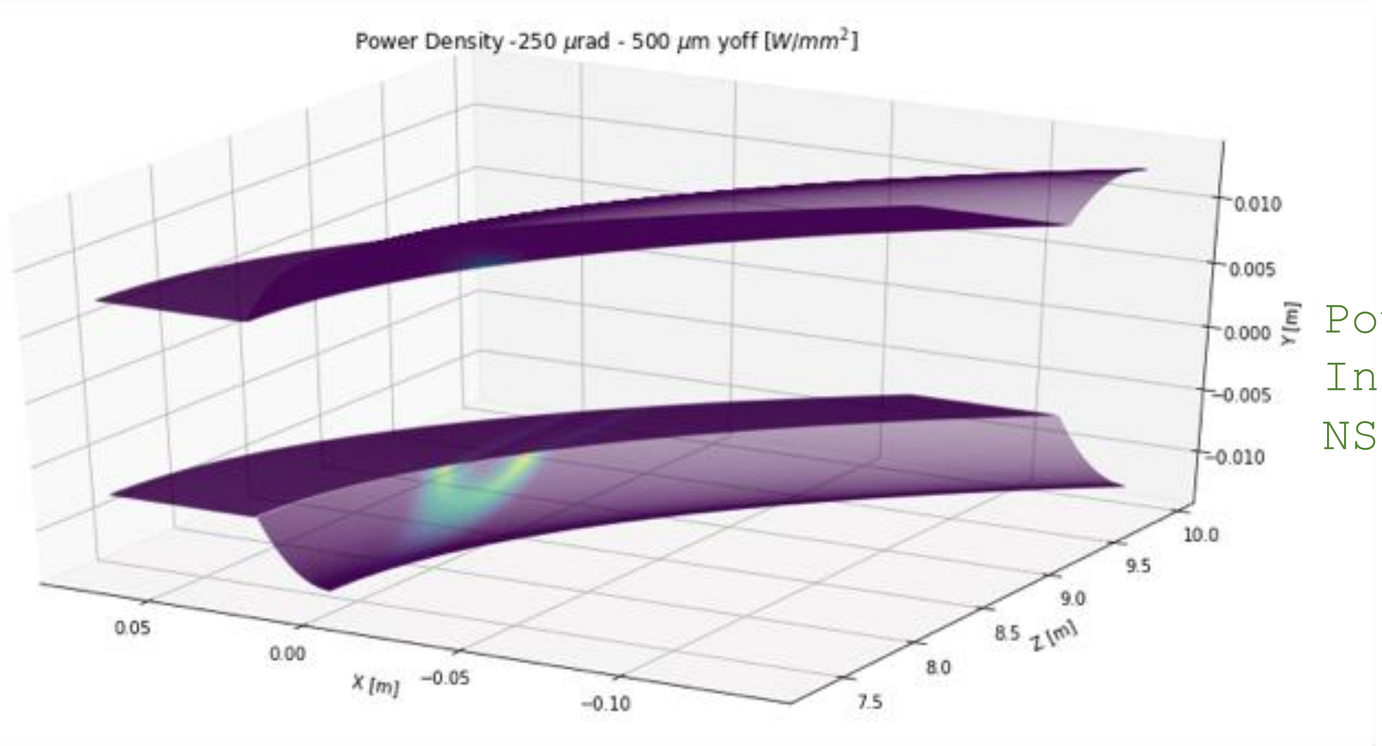

```
rotations=[0, osr.pi()/3, 0]
```




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

# Power Density – Parametric 3D

- Advantage: Much easier visualization




Power Density
Inner wall,
NSLS-II Dipole chamber

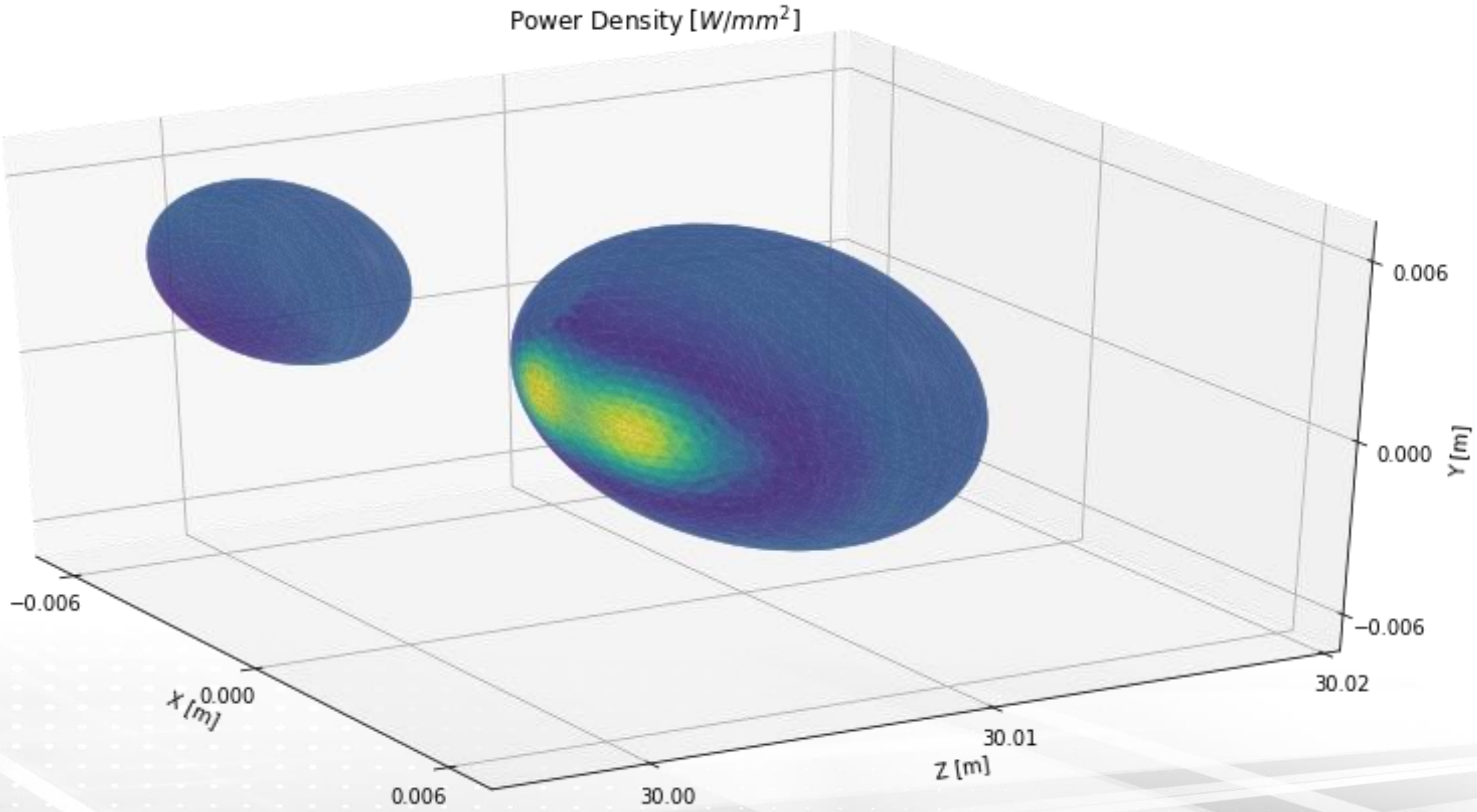
- Major disadvantage: Very difficult for complex objects

U.S. DEPARTMENT OF ENERGY

BROOKHAVEN NATIONAL LABORATORY

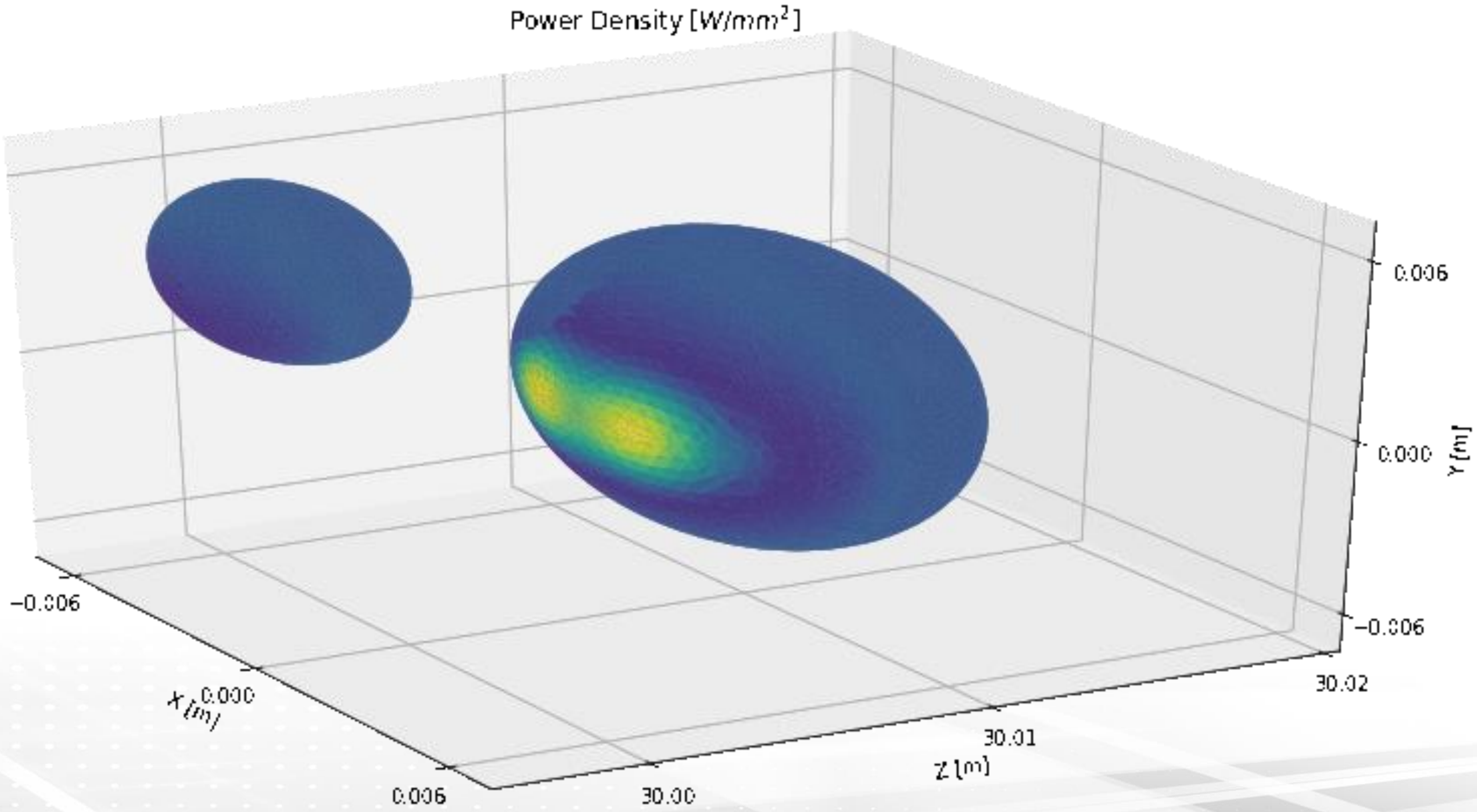
# Power Density CAD

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



# Power Density CAD

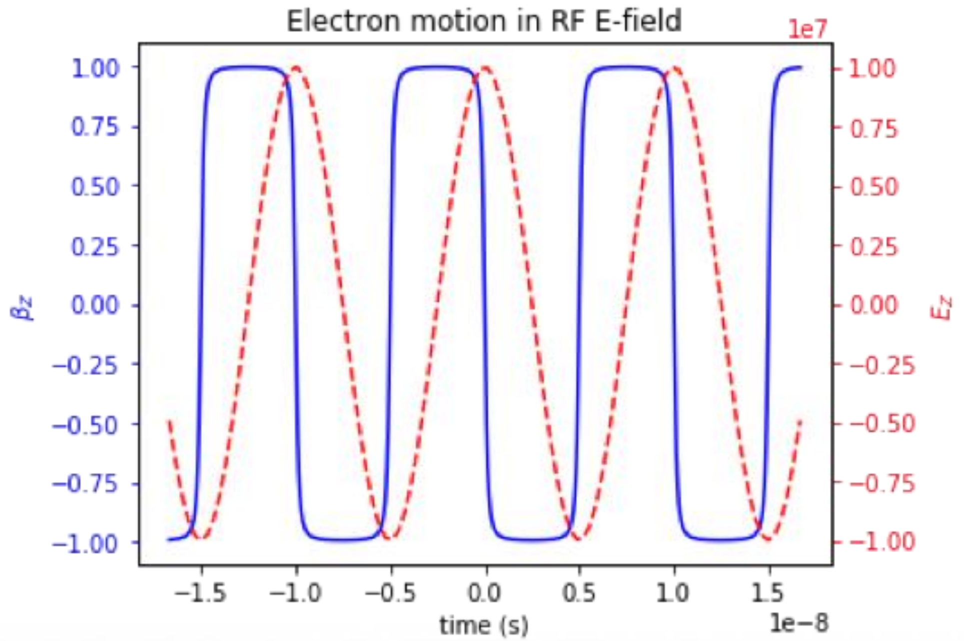
- Can now import CAD model in STL format
- Allows for substantially more complex objects (than shown here)

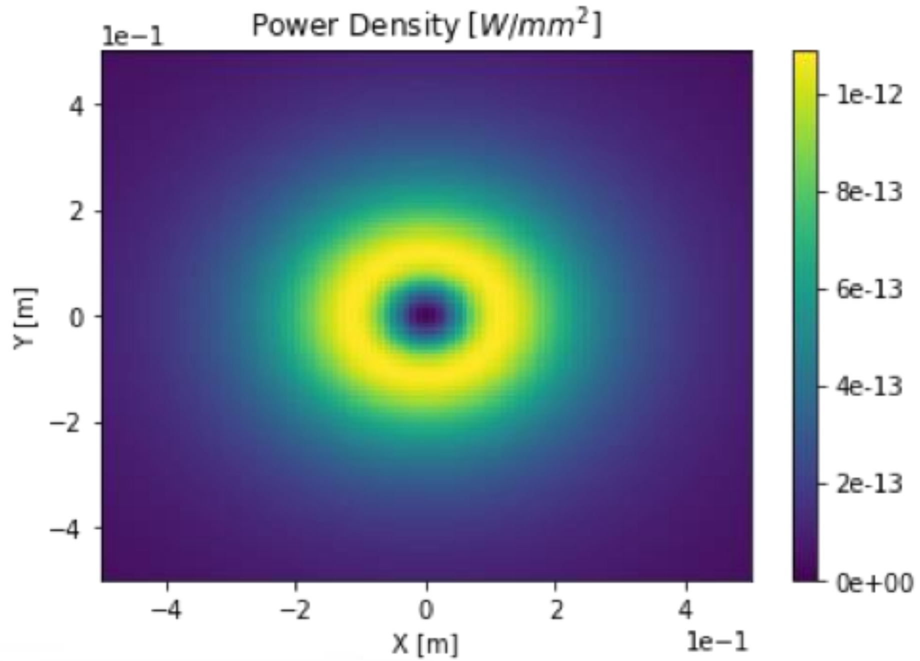


# Time Dependence

- 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**