

Simulating Huygens' Surfaces

Yankun (Alex) Meng



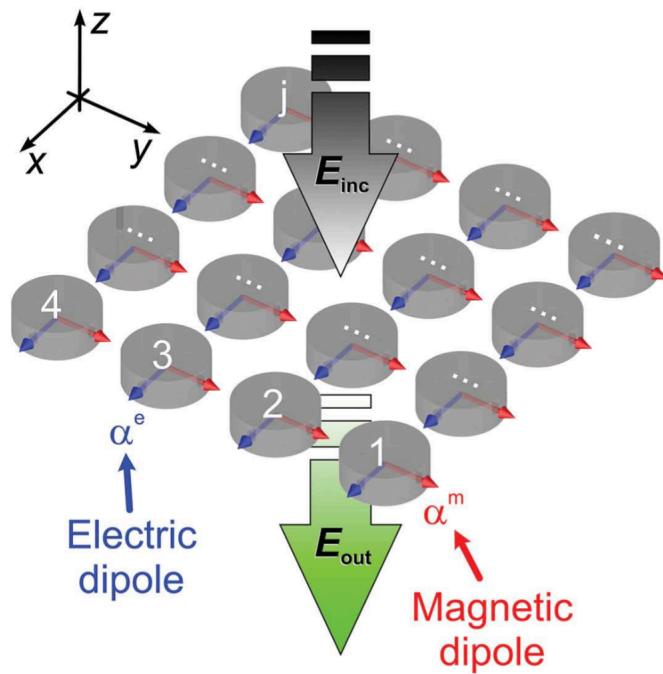


Figure 1: Huygens Metasurface

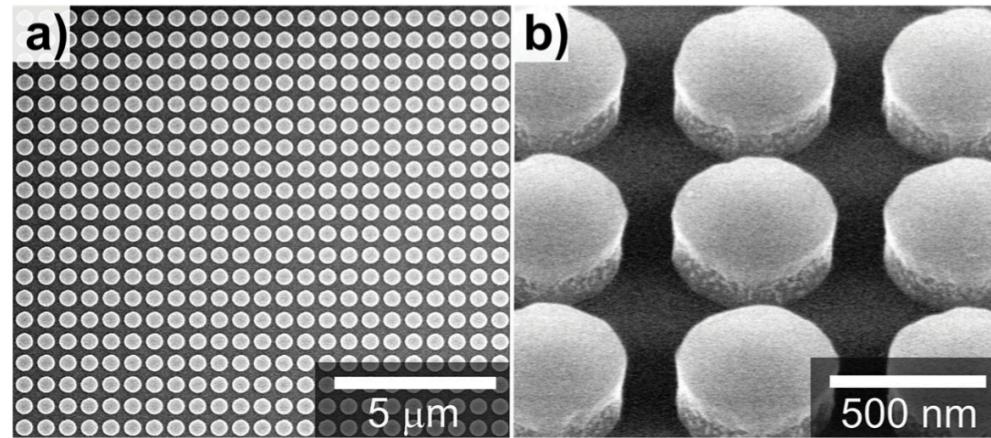


Figure 3. a) Top-view and b) side-view electron micrographs of a typical silicon-nanodisk sample with a disk radius of $r_d \approx 242$ nm, a disk height of $h = 220$ nm, and a lattice constant of $a \approx 666$ nm.

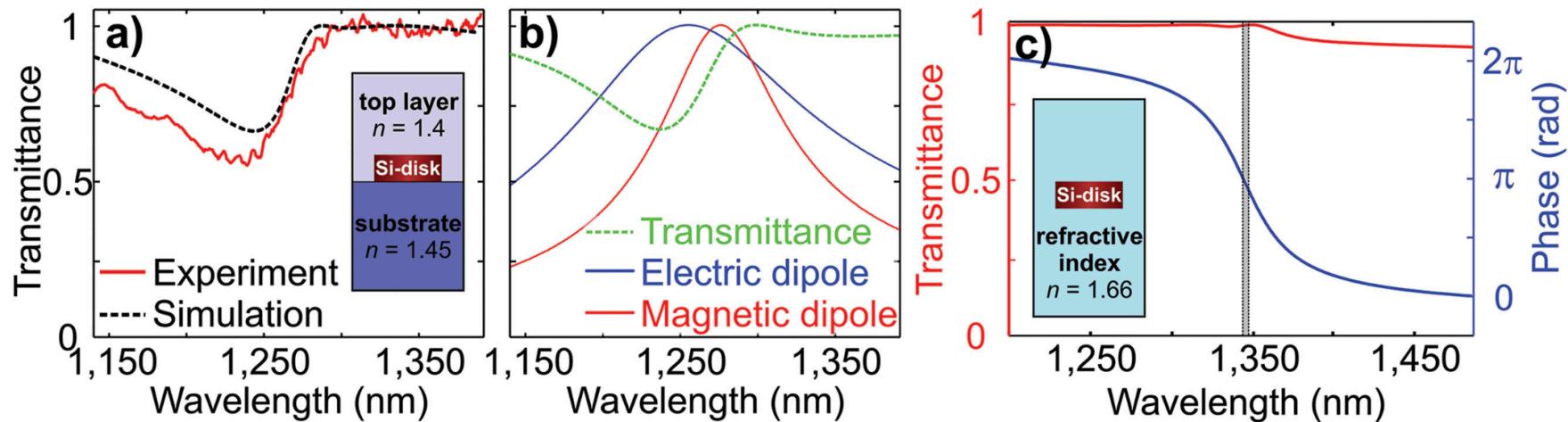


Figure 5. a) Experimental transmittance spectrum of the Huygens' metasurface [overlap case shown in Figure 4b)] embedded in spin-on glass ($n=1.4$) after handle-wafer removal (red-solid line) and corresponding numerical calculation (black-dashed line). b) Transmittance intensity (green line) gained from the analytical model and its decomposition into electric (blue) and magnetic (red) dipole contributions. The parameters for the electric and magnetic dipole response are: $\lambda_{\text{mag}} = 1275 \text{ nm}$, $\gamma_{\text{mag}} = 2\pi 6.8 \text{ THz}$, and $\lambda_{\text{el}} = 1251 \text{ nm}$, $\gamma_{\text{el}} = 2\pi 14.1 \text{ THz}$. c) Numerically calculated transmittance for a Huygens' metasurface ($r_d = 242 \text{ nm}$, $h = 220 \text{ nm}$) embedded in a homogeneous medium with optimized refractive index ($n = 1.66$). The transmittance intensity (red solid line) and transmittance phase (blue solid line) for resonance overlap at $\lambda_{\text{el},n=1.66} \approx \lambda_{\text{mag},n=1.66} \approx 1340 \text{ nm}$ feature a transmission above $T_{\min} \approx 0.995$ and full phase coverage of 2π .

In this notebook, the Huygens' Metasurface figure 5(a) and 5(c) are reproduced, and mesh study is done for the transmittance. [Link to Paper](#)

Simulation Overview

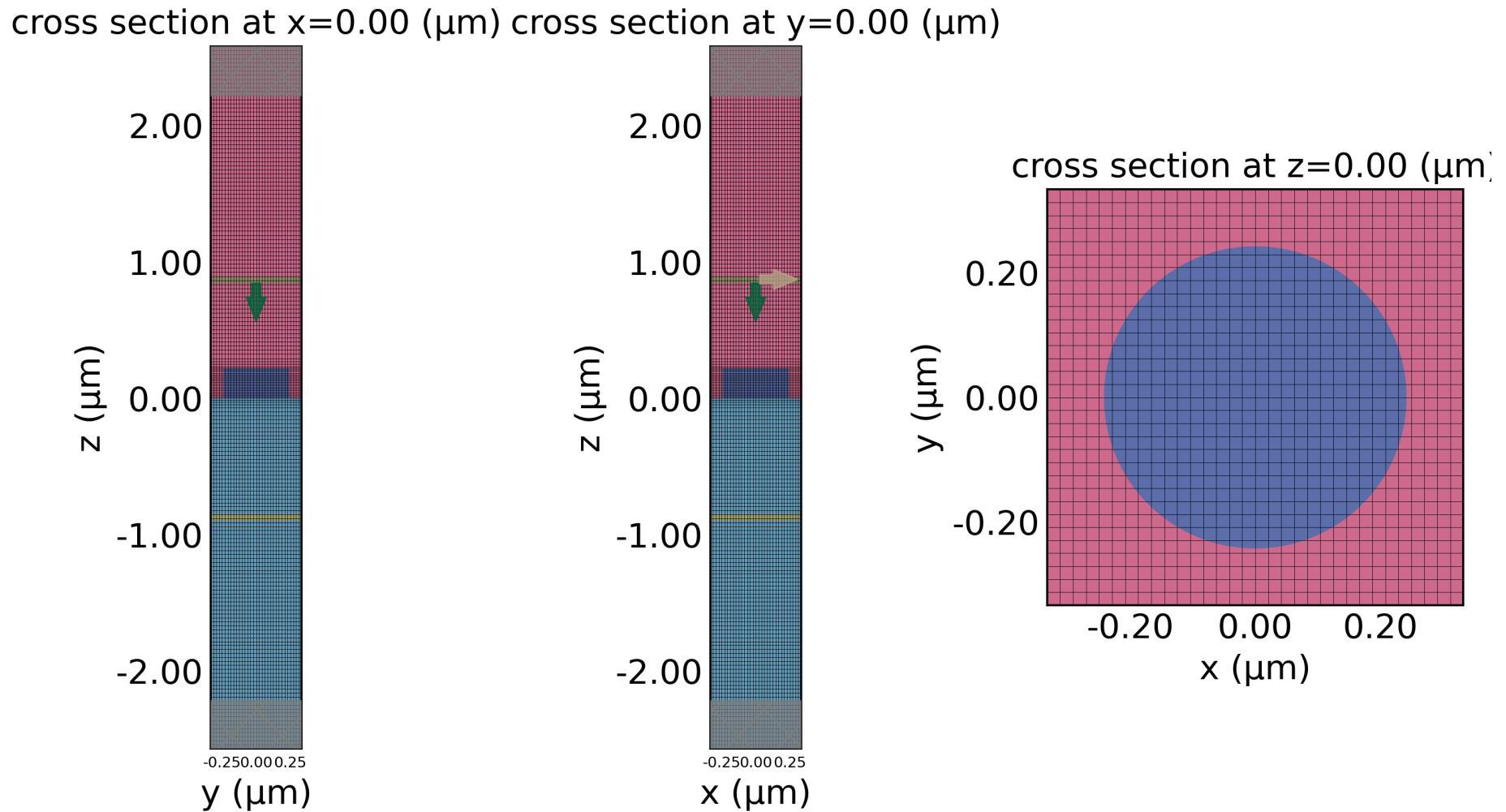
```
1 # Import the necessary packages
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import tidy3d as td
5 import tidy3d.web as web
6 import scienceplots
7
8 # Set logging level to ERROR to reduce output verbosity
9 td.config.logging_level = "ERROR"
```

- For the transmittance and phase, two different background materials were used based in the paper, so two separate simulations were ran (with background medium as the only difference).

Note: Several other technical variables need to be changed. In tidy3D simulation, transmittance should be measured with `td.FluxMonitor` and phase should be measured with `td.FieldMonitor`. The `run_time` also needs to increase since the background medium has a higher



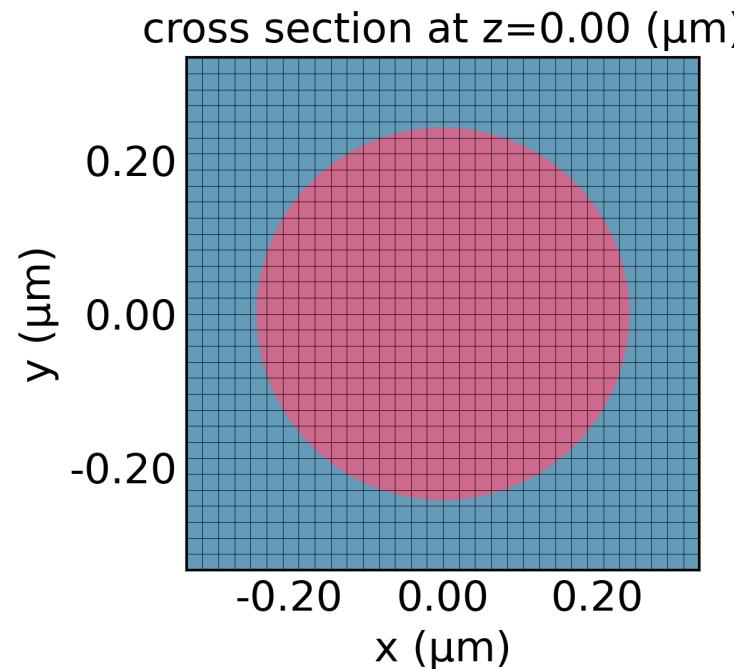
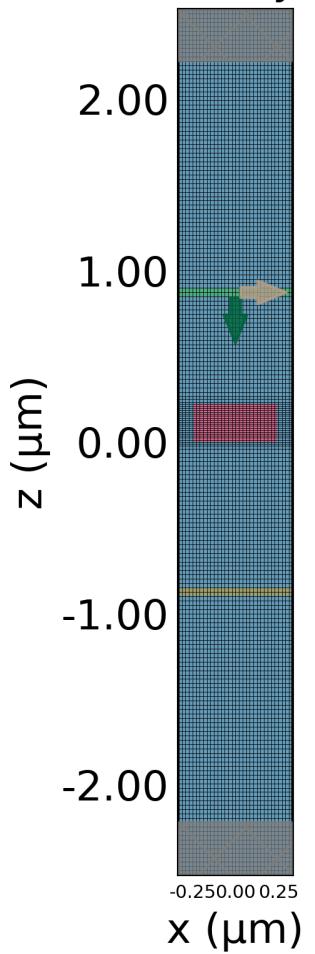
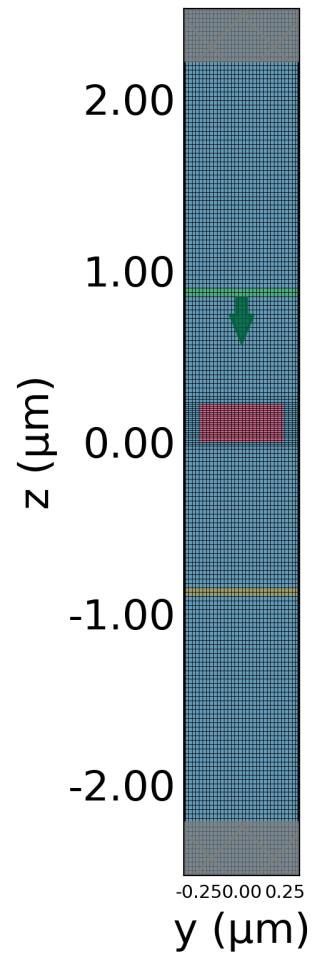
Simulation 1



Top Layer $n=1.4$, Bottom Layer $n=1.45$

Simulation 2

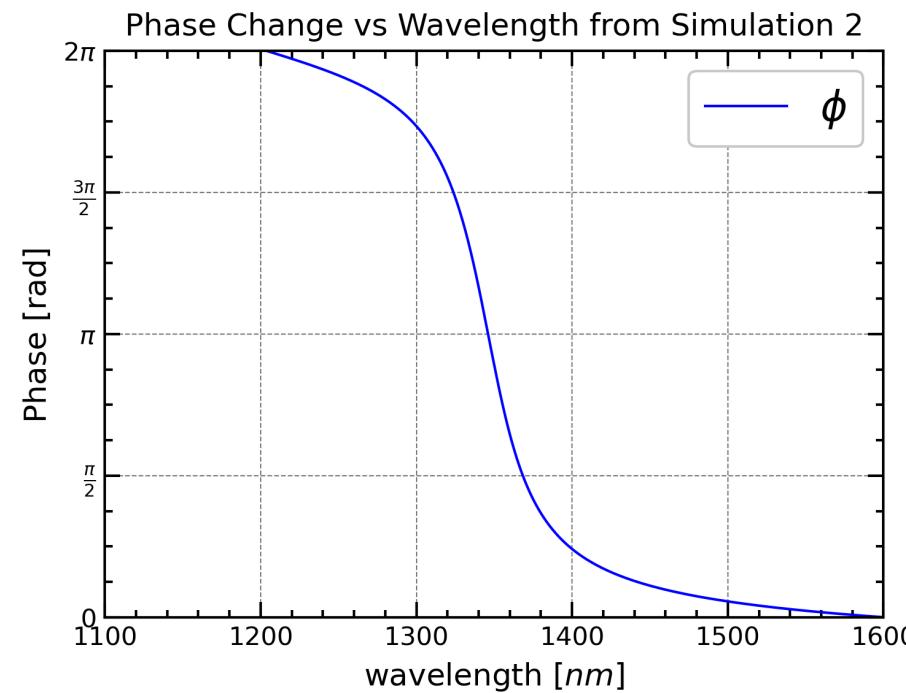
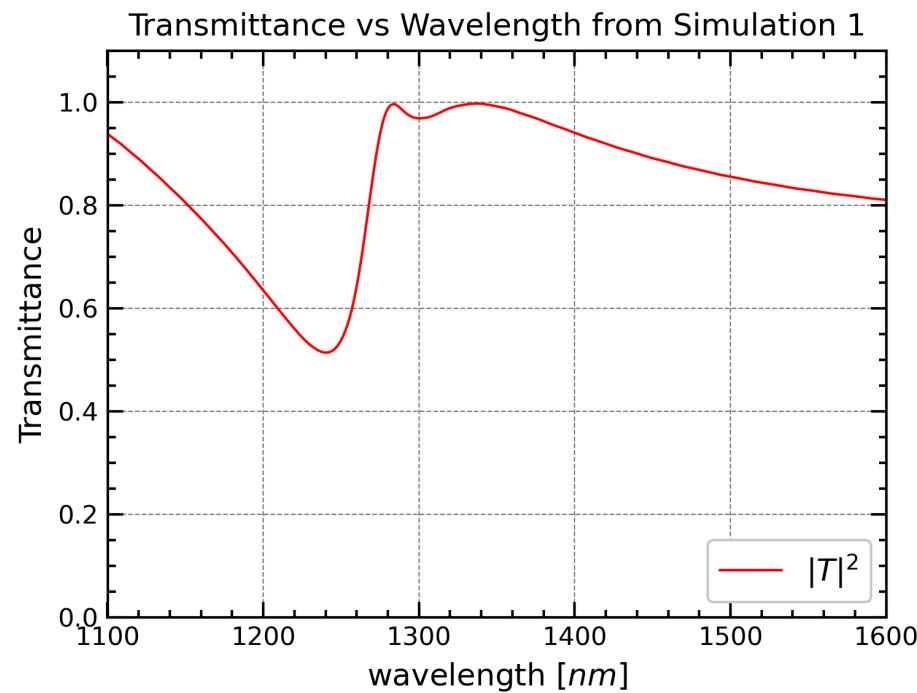
cross section at $x=0.00$ (μm) cross section at $y=0.00$ (μm)



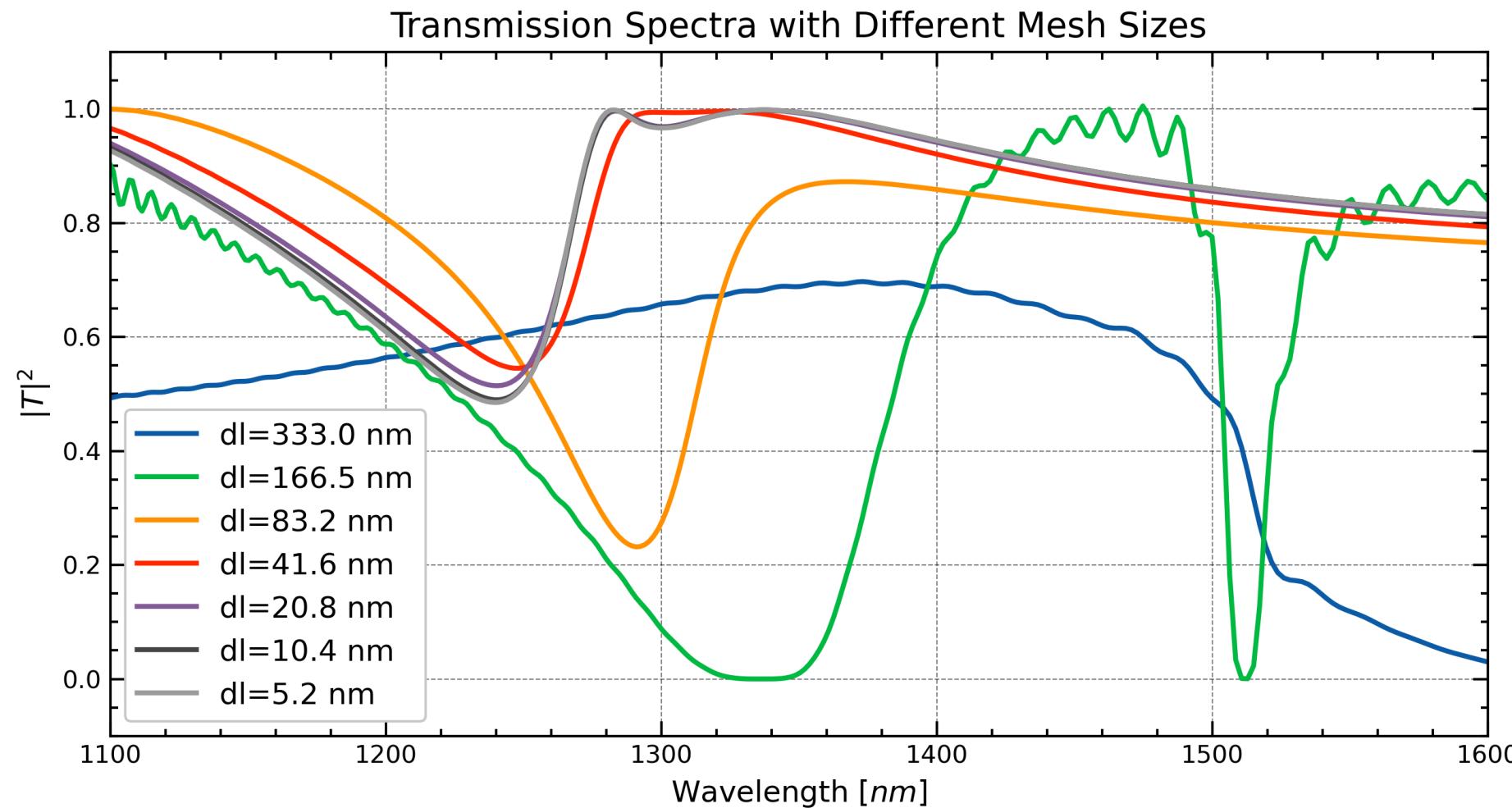
Background medium $n=1.66$



Simulation Results



Mesh Study Results



Convergence of Transmittance



Initialization

Here we follow the seven steps of initialization I wrote down in the tutorial:

0. Frequency Range Specification
1. Computational Domain Size
2. Grid Specifications (Discretization size)
3. Structures and Materials
4. Sources
5. Monitors
6. Run time
7. Boundary Condition Specification

o Frequency Range Specification

```
1 # 0 Define a FreqRange object with desired wavelengths
2 fr = td.FreqRange.from_wvl_interval(wvl_min=1.1, wvl_max=1.6)
3 N = 301 # num_points
4 freq0 = fr.freq0
5 lda0 = td.C_0 / fr.freq0
```



1 Computational Domain Size

```
1 # 1 Computational Domain Size
2 h = 0.220 # Height of cylinder
3 spc = 2
4 Lz = spc + h + h + spc
5
6 Px = Py = P = 0.666 # periodicity
7 sim_size = [Px, Py, Lz]
```



2 Grid Resolution

Grid resolution is uniform grid in the horizontal direction with a yee cell length of $\frac{P}{32}$ where P is the periodicity. In the vertical direction, [AutoGrid](#) means it's non-uniform and adjusted based on the wavelength of the particular medium. Here, `min_steps_per_wvl=32` means we are taking a minimum of 32 steps based on the wavelength, which will be shorter in the medium with a higher index of refraction.

```
1 # 2 Grid Resolution
2 dl = P / 32
3 horizontal_grid = td.UniformGrid(dl=dl)
4 vertical_grid = td.AutoGrid(min_steps_per_wvl=32)
5 grid_spec=td.GridSpec(
6     grid_x=horizontal_grid,
7     grid_y=horizontal_grid,
8     grid_z=vertical_grid,
9 )
```



3 Structures and Materials

Structures and Materials for the meta-atom

```
1 r = 0.242 # radius of the cylinder
2 n_Si = 3.5
3 Si = td.Medium(permittivity=n_Si**2, name='Si')
4 cylinder = td.Structure(
5     geometry=td.Cylinder(center=[0, 0, h / 2], radius=r, length=h, axis=2), medium=Si
6 )
```

Background Medium for Figure 5(a) ($n_1 = 1.4, n_2 = 1.45$)

```
1 # Background medium for the first simulation
2 n_glass = 1.4
3 n_SiO2 = 1.45
4 glass = td.Medium(permittivity=n_glass**2, name='glass')
5 SiO2 = td.Medium(permittivity=n_SiO2**2, name='oxide')
6
7 substrate = td.Structure(
8     geometry=td.Box(
9         center=(0,0,-Lz/2),
10        size=(td.inf,td.inf,2 * (spc+h))
11    ),
12    medium=SiO2,
13    name='substrate'
14 )
15
```



Background Medium for Figure 5(c) ($n = 1.66$)

```
1 # Background medium for the second simulation
2 # Polymer
3 n_polymer = 1.66
4 polymer = td.Structure(
5     geometry=td.Box(
6         center=(0,0,0),
7         size=(td.inf,td.inf,td.inf)
8     ),
9     medium=td.Medium(permittivity=n_polymer**2, name='polymer'),
10    name='polymer'
11 )
```



4 The Source

The source is a simple Plane wave that traverses in the -z axis, placed $\frac{\lambda_0}{2}$ distance above the metaatom in the computational domain.

Polarization is along the x-axis, that's what `pol_angle=0` means.

```
1 source = td.Planewave(  
2     source_time=fr.to_gaussian_pulse(),  
3     size=(td.inf, td.inf, 0),  
4     center=(0, 0, Lz/2 - spc + 0.5 * lda0),  
5     direction="-",  
6     pol_angle=0  
7 )
```



5 Monitors

Monitor for Transmittance

```
1 flux_monitor = td.FluxMonitor(  
2     center=(0, 0, -Lz/2 + spc - 0.5 * lda0),  
3     size=(td.inf, td.inf, 0),  
4     freqs=fr.freqs(N),  
5     name="flux_monitor"  
6 )
```

Monitor for Phase

```
1 # We use FieldMonitor instead of DiffractionMonitor because  
2 # DiffractionMonitor only gives you amplitudes of diffraction orders,  
3 # losing phase detail if you care about continuous phase.  
4 field_monitor = td.FieldMonitor(  
5     center=(0, 0, -Lz/2 + spc - 0.5 * lda0),  
6     size=(td.inf, td.inf, 0),  
7     fields=["Ex"],  
8     freqs=fr.freqs(N),  
9     name="field_monitor"  
10 )
```



6 Run Time

```
1 bandwidth = fr.fmax - fr.fmin  
2 run_time_short = 50 / bandwidth # run_time for the transmittance simulation  
3 run_time_long = 200 / bandwidth # run_time for the phase simulation
```



7 Boundary Conditions

We apply PML in the +Z and -Z surfaces.

```
1 bc = td.BoundarySpec(  
2     x=td.Boundary.periodic(),  
3     y=td.Boundary.periodic(),  
4     z=td.Boundary.pml()  
5 )
```



Helper Function for simulation

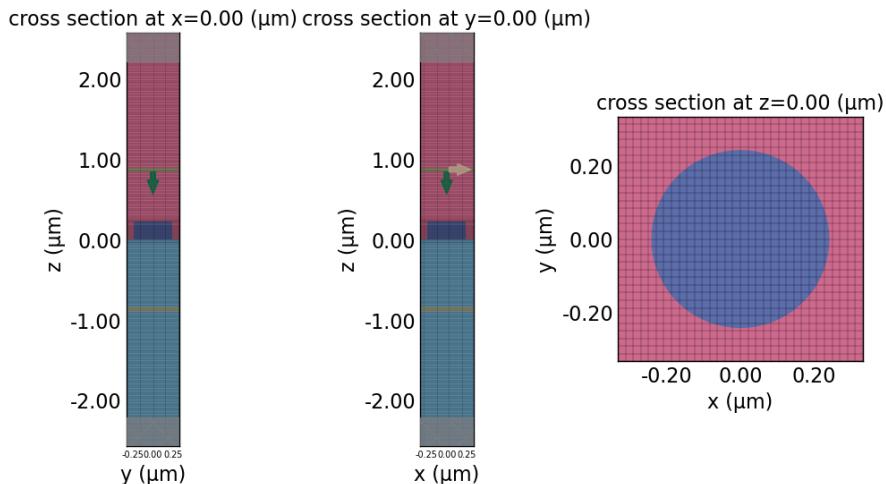
Since we have to run simulation two times, it is convenient to abstract out what are the differences to the two simulations and make defining simulations easier. Always follow the **DRY Principle**.

```
1 def simulation_helper(background, monitors, run_time):
2     sim_empty=td.Simulation(
3         size=sim_size,
4         grid_spec=grid_spec,
5         structures=background,
6         sources=[source],
7         monitors=monitors,
8         run_time=run_time,
9         boundary_spec=bc
10    )
11
12    background.append(cylinder)
13    sim_actual = td.Simulation(
14        size=sim_size,
15        grid_spec=grid_spec,
16        structures=background,
17        sources=[source],
18        monitors=monitors,
19        run_time=run_time,
20        boundary_spec=bc
21    )
```



Transmittance Simulation

```
1 sims = simulation_helper(  
2     background=[substrate, glass],  
3     monitors=[flux_monitor],  
4     run_time=run_time_short  
5 )
```



```
1 batch = web.Batch(simulations=sims, verbose=True)  
2 batch_data = batch.run(path_dir="data/huygens5a")
```

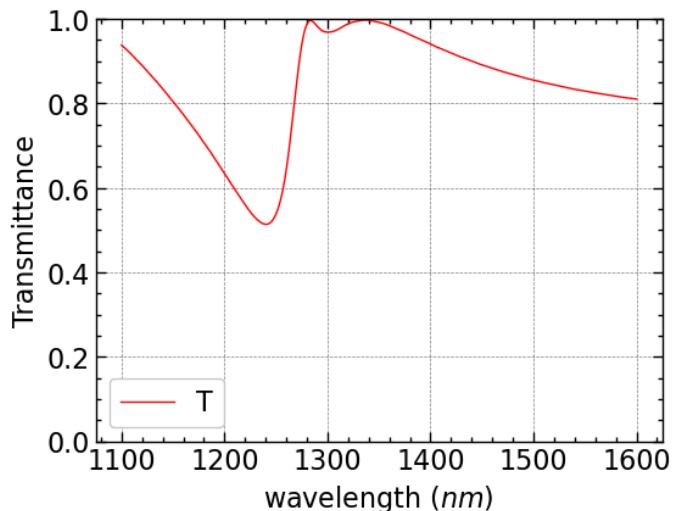
```
02:20:56 EDT Started working on Batch containing 2 tasks.  
02:20:57 EDT Maximum FlexCredit cost: 0.050 for the whole batch.  
Use 'Batch.real_cost()' to get the billed FlexCredit cost after the  
Batch has completed.  
02:20:58 EDT Batch complete.
```



Transmittance Results

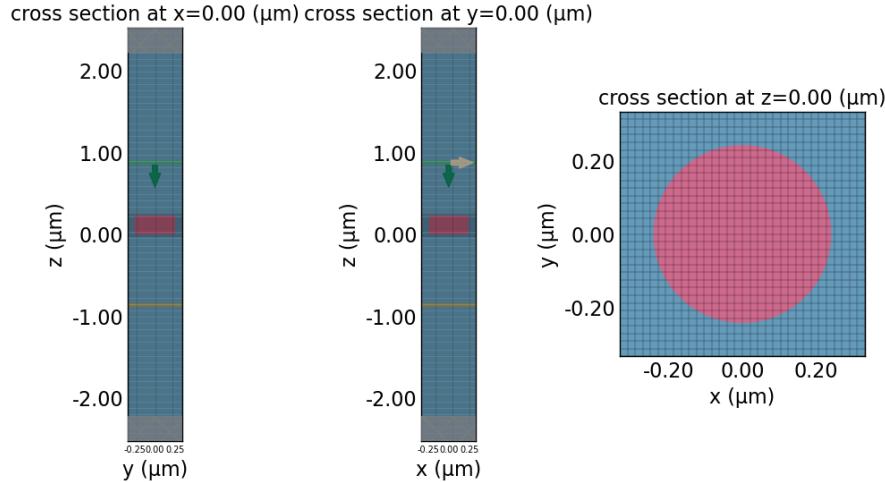
```
1 # this uses scienceplots to make plots look better
2 plt.style.use(['science', 'notebook', 'grid'])
3 T = batch_data["actual"]["flux_monitor"].flux / batch_data["norm"]["flux_monitor"].flux

1 # plot transmission, compare to paper results, look similar
2 fig, ax = plt.subplots(1, 1, figsize=(6, 4.5))
3 plt.plot(td.C_0 / fr.freqs(N) * 1000, np.abs(T)**2, "r", lw=1, label="T")
4 plt.xlabel(r"wavelength ($nm$)")
5 plt.ylabel("Transmittance")
6 plt.ylim(0, 1)
7 plt.legend()
8 plt.show()
```



Phase Simulation

```
1 sims = simulation_helper(  
2     background=[polymer],  
3     monitors=[field_monitor],  
4     run_time=run_time_long  
5 )
```



```
1 batch = web.Batch(simulations=sims, verbose=True)  
2 batch_data = batch.run(path_dir="data/huygens5c")
```

```
02:21:11 EDT Started working on Batch containing 2 tasks.  
02:21:13 EDT Maximum FlexCredit cost: 0.050 for the whole batch.  
Use 'Batch.real_cost()' to get the billed FlexCredit cost after the  
Batch has completed.  
02:21:14 EDT Batch complete.
```



Phase Results

```
1 # Data Extraction
2 Ex_actual = batch_data["actual"]["field_monitor"].Ex
3 Ex_norm = batch_data["norm"]["field_monitor"].Ex
4 Ex = Ex_actual / Ex_norm

1 # 1. Compute average over the xy-plane
2 Ex_avg = np.mean(Ex[:, :, 0, :], axis=(0,1))
3
4 # 2. Compute phase
5 phase_avg = np.angle(Ex_avg)
6
7 # 3. Unwrap phase to remove ±pi jumps
8 phase_avg_unwrapped = np.unwrap(phase_avg)
9
10 # 4. Make relative to first point (optional)
11 phase_rel = phase_avg_unwrapped - phase_avg_unwrapped[0]
12
13 phase_actual = np.unwrap(np.angle(np.mean(Ex_actual[:, :, 0, :], axis=(0,1))))
14 phase_norm = np.unwrap(np.angle(np.mean(Ex_norm[:, :, 0, :], axis=(0,1))))
```



```
1 fig, ax = plt.subplots(1, 1, figsize=(6, 4.5))
2 plt.plot(td.C_0 / fr.freqs(N) * 1000, phase_rel, "r", lw=1, label="$\phi$")
3 plt.plot(td.C_0 / fr.freqs(N) * 1000, phase_actual, "b", lw=1, label="Actual $\phi$")
4 plt.plot(td.C_0 / fr.freqs(N) * 1000, phase_norm, "g", lw=1, label="Norm $\phi$")
5 plt.xlabel(r"wavelength ($nm$)")
6 plt.ylabel("Phase")
7 plt.legend()
```



Final Plotting

```
1 fig, axes = plt.subplots(1, 2, figsize=(12, 4))
2
3 # work on the first figure
4 ax = axes[0]
5 ax.tick_params(axis="both", labelsize=10)
6 ax.plot(td.C_0 / fr.freqs(N) * 1000, np.abs(T)**2, "r", lw=1, label="$|T|^2$")
7 ax.set_xlabel(r"wavelength [$nm$]", fontsize=12)
8 ax.set_ylabel("Transmittance", fontsize=12)
9 ax.set_title("Transmittance vs Wavelength from Simulation 1", fontsize=12)
10 ax.set_xlim(1100, 1600)
11 ax.set_ylim(0, 1.1)
12 ax.legend(loc="lower right", fontsize=12)
13
14 # work on the second figure
15 ax = axes[1]
16 ax.tick_params(axis="both", labelsize=10)
17 ax.plot(td.C_0 / fr.freqs(N) * 1000, phase_rel, "b", lw=1, label="$\phi$")
18 ax.set_xlabel(r"wavelength [$nm$]", fontsize=12)
19 ax.set_ylabel("Phase [rad]", fontsize=12)
20 ax.set_title("Phase Change vs Wavelength from Simulation 2", fontsize=12)
21 ax.set_xlim(1100, 1600)
22 ax.set_ylim(0, np.pi*2)
23 yticks = [0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi]
24 ytick_labels = [r"$0$", r"$\frac{\pi}{2}$", r"$\pi$",
25                 r"$\frac{3\pi}{2}$", r"$2\pi$"]
26 ax.set_yticks(yticks)
27 ax.set_yticklabels(ytick_labels)
```



Mesh Study

Here, we set out to study the effect of different yee cell length on the transmittance.

```
1 dls = [P/2, P/4, P/8, P/16, P/32, P/64, P/128] # mesh study list
2 sims = {}

1 # for each dl in dls
2 for i, dl in enumerate(dls):
3     # 2 Grid Specifications
4     horizontal_grid = td.UniformGrid(dl=dl)
5     vertical_grid = td.AutoGrid(min_steps_per_wvl=32)
6     grid_spec=td.GridSpec(
7         grid_x=horizontal_grid,
8         grid_y=horizontal_grid,
9         grid_z=vertical_grid,
10    )
11
12     # 4 Sources
13     source = td.Planewave(
14         source_time=fr.to_gaussian_pulse(),
15         size=(td.inf, td.inf, 0),
16         center=(0, 0, Lz/2 - spc + 2 * dl),
17         direction="-",
18         pol_angle=0
19    )
20
```



Mesh Study Results

```
1 # Extract results
2 x = td.C_0 / fr.freqs(N) * 1000
3 Ts = []
4 for i in range(len(dls)):
5     Ts.append(batch_data[f"actual{i}"]["flux"].flux / batch_data[f"norm{i}"]["flux"].flu
6
7 # Plot results
8 plt.figure(figsize=(10, 5))
9 for i, T in enumerate(Ts):
10     plt.plot(x, np.abs(T)**2, "--", lw=1, label=f"dl={dls[i] * 1000:.1f} nm")
11 plt.xlabel(r"Wavelength [nm]", fontsize=12)
12 plt.ylabel(r"$|T|^2$ ", fontsize=12)
13 plt.xlim(1100, 1600)
14 plt.ylim(-0.1, 1.1)
15 plt.legend(fontsize=12)
16 plt.tick_params(axis='both', labelsize=10) # change tick label size to 10
17 plt.title("Transmission Spectra with Different Mesh Sizes", fontsize=14)
18 plt.savefig("mesh_convergence.png", dpi=300)
19 plt.show()
```

