School of Computer Science and Statistics

Biomimicry in Anti-Reflective Moth eyes for Solar Panels via MEEP FDTD software

Written by

Anirudh Kundu

under the supervision of **Dr Fergal Shevlin**, and submitted to the School of Computer Science and Statistics in partial fulfilment of the requirements for the degree of

BA (Mod) Computer Science

at Trinity College Dublin, The University of Dublin.

Acknowledgements

I would like to thank my supervisor, Dr Fergal Shevlin, for his dedicated support and guidance. Dr Fergal continuously provided encouragement and was always willing to guide and encourage me towards the very insightful project. Most importantly, I am grateful for my family's unequivocal, and loving support throughout my final year, and Trinity College for giving me an opportunity to learn and grow under its guidance.

Contents

Abstract

Solar panels have been used as an alternative green energy source since 1883. Solar panels do not yet have ideal efficiencies. One major factor in the loss of energy in a solar cell is the presence of a thin film or a substrate where the loss of light energy is due to reflection due to the high value of the refractive index of the substrate. Due to years of evolution, the eyes of some insects have evolved complex eye apparatus that aid in absorbing all the light from surroundings, rendering no loss of light. The eyes of the tiny insect Moth have small patterns and protuberances, that have shown to decrease the reflection and thus increase the efficiency of transmittance of light. This paper aims to apply the previously proposed optimized models of the Moth eye structure via different methods, apply the proposed models and prove its effectiveness and efficiency over traditional solar panel films, via running high accuracy FDTD based simulations in open-source software called MEEP, an MIT based photonics software developed in 2006.

1 Introduction

The invention and production of solar panels in society was plausibly an effective way to diminish the dependency on fossil fuels reducing the carbon footprint and mitigating global warming by lowering the emission of greenhouse gases.

The solar cells in the solar panels are not directly in contact with the environment.

Solar panels have a transparent layer over the solar cells, to prevent them from natural effects, such as rain, wind, dust, and animals.

However, there is a significant energy loss in the transfer of energy. Among the total loss of energy, a significant 4-5% of energy is lost due to the reflection of light from the protecting layer, as no surface transmits all the light rays. This phenomenon is understandable by Fermat's Principle which states- ‘the path taken between two points by a light ray is the path transverse in the least time,’ thus explaining the properties of light rays reflected off glass and other such objects.

Since the first use of Biomimicry in Research in 1950 by American biophysicist and polymath Otto Schmitt developed the concept of “biomimetics”, Where his doctoral research, consisted of a Schmitt trigger by studying the nerves in squid, attempting to engineer a device that replicated the biological system of nerve propagation. This report uses biomimicry to test the problem.

Biomimicry is an approach to innovation that seeks sustainable solutions to challenges by emulating nature's time-tested patterns and strategies. In short, biomimicry takes the innovations that exist in nature and applies them to solve present simple or complex problems.

Thus, here we use Biomimicry of the Moth Eye to try to tackle the problem of loss of energy due to reflection. Moths have unique sub-wavelength structures coating their eyes which have proven to dramatically minimise light reflection over a much broader range of wavelengths than conventional coatings.

A moth's eye is coated with a regular pattern of conical protuberances called a corneal nipple array, 200-300 nanometers in height and spacing that gradually bends incoming light. The light waves interfere with one another rendering a near-zero reflection of light.

Thus, in this report, we will test and discuss if Moth eye based anti-reflective structures can be a viable solution to the loss of energy due to reflection in solar panels.

To test the corneal nipple array which is in order of a few nanometers, simulation of the structure under various wavelengths is necessary. Thus, photonics software that is highly accurate and efficient in nano-scale calculations is used, to test the corneal structure.

In this report, we use an Open-source software photonics software i.e., MEEP, for accurate calculations in the sub nanoscale domain.

1.1 Contribution.

- Show the benefits of moth-eye-like structures, their specific structure, and the properties of the structure.
- Calculate the exact increase in efficiency of the corneal array on the solar panels, between 400nm - 800nm wavelength of light.

2 Moth Eye Structure

In the first chapter, the morphology of the actual Moth eye structure is first described and then the specific key necessary components of the Moth eye which are pre-modelled are defined. Towards the end of the chapter, the relevant proposed structures are finalised that can be further simulated in MEEP

External View of the Moth Ey

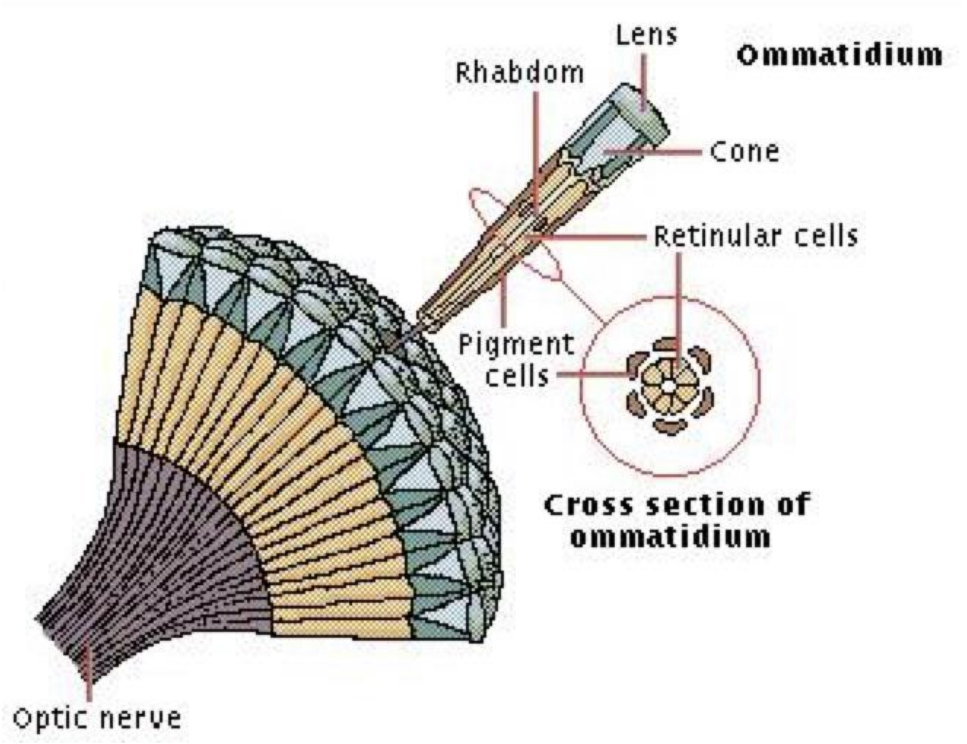
2.1 Morphology of the Complex Moth Eye

Light is perceived by insects through several different receptors. Most adult insects have a pair of compound eyes. Compound eyes are so-called because they are constructed from many simpler similar units called ommatidia.

Each ommatidium consists of an optical, light-gathering part and a sensory part, which transforms light into electrical energy.

The exact morphology of compound eyes of different flies differs across different insects in the class ‘Insecta’ in the phylum Arthropod, in the Animal Kingdom.

Ommatidia are tiny independent photoreception units that consist of a cornea, lens, and photoreceptor cells that distinguish brightness and colour.

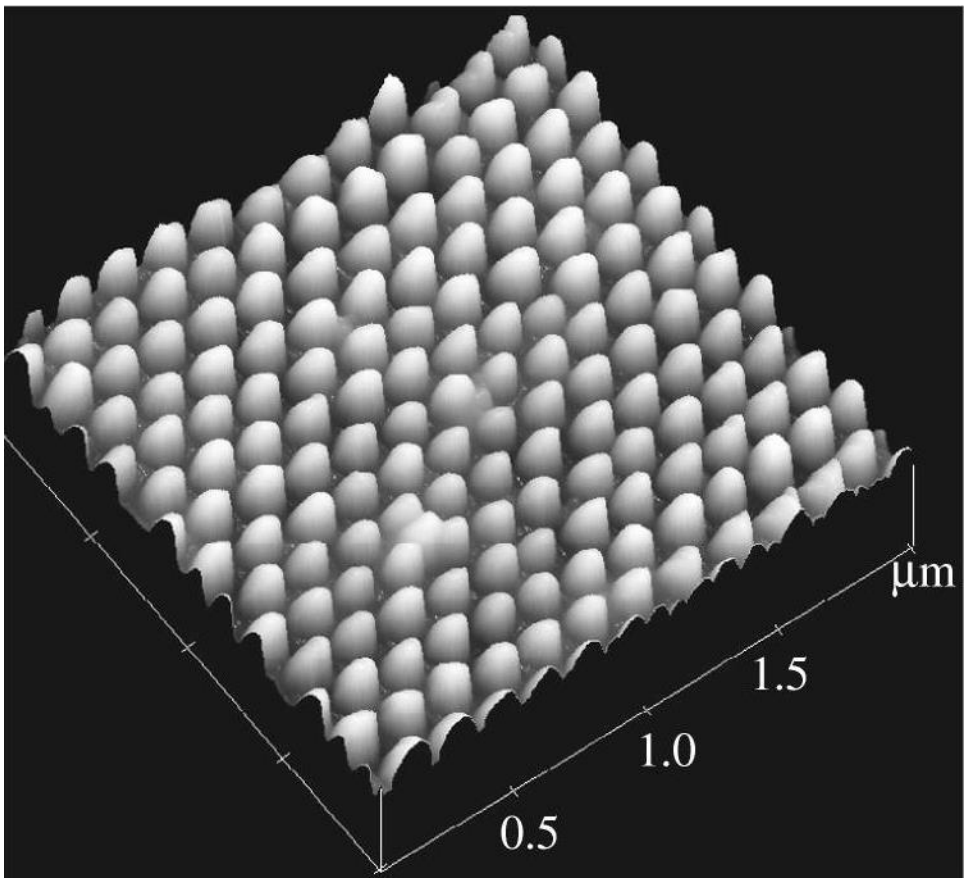


View of a single Ommatidia

2.2 Corneal Array Structure' of the Moth Eye

In this report, only the light-gathering part, the Lens of the ommatidia i.e., the cornea is introspected. The ommatidia's outer surface structure and morphology are called the corneal structure and thus will be referred to as the 'corneal structure' in this report.

The outer surfaces of moth corneal lenses are covered with a regular pattern of conical protuberances, 200-300 nm in height and spacing. These protuberances are shown to reduce light reflection by creating a refractive index gradient between the air-lens interface, gradually transitioning the change in light speed between the air and eye and hence minimising reflection.



Microscopic Image of the Moth Cornea

This evolution of corneal nipples in moth-like insects can be explained by.

corneal nipples function to reduce the eye glare of moths that are inactive during the day, so to make them less visible to predators.

2.3 Optimised RCWA structures

Presently there exist several optimisations of moth-like structures in development called Anti-Reflective-Microstructures, ARM.

ARM are derived variants or improved versions of the corneal structure, for proposed applications in distinctive domains.

The ARM are simulated by a process called the RCWA method.

Rigorous coupled-wave analysis (RCWA) is a semi-analytical method in computational electromagnetics that is most typically applied to solve scattering from periodic dielectric structures.

However, in this report structures simulated via RCWA are used as a reference, then implemented and simulated in MEEP.

Structure 1.

In this report, two similar RCWA based ARM are described and simulated.

The first structure simulated in the simulation is the SWS (Sub Wavelength Structure).

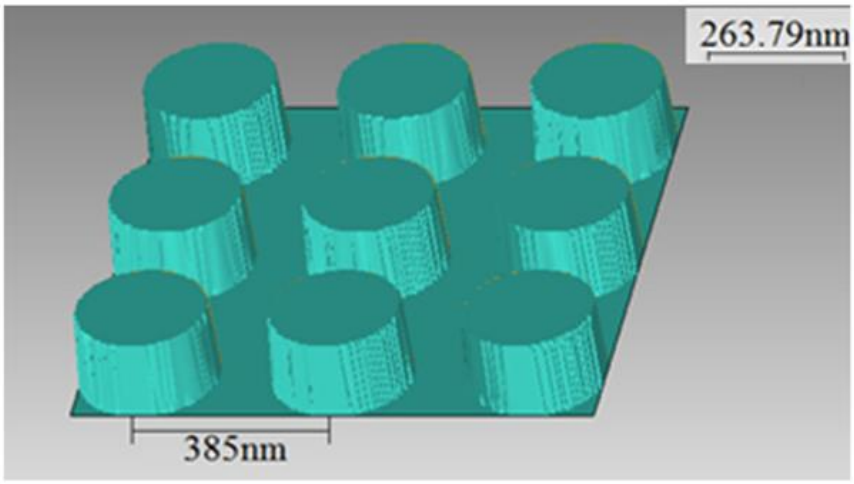


Figure 1: SWS Structure

Items	Patterned Height (nm)	Periodic (nm)	Bottom Diameter (nm)	Top Diameter (nm)
Pillar	160	385	270	265

Since SWS structures resemble pillars; this structure is called a 3 x 3 Pillar structure. The accurate dimensions are in Figure 1.

Structure 2.

The second RCWA based ARM resembles the Moth compound eye, corneal nipple array more accurately is simulated.

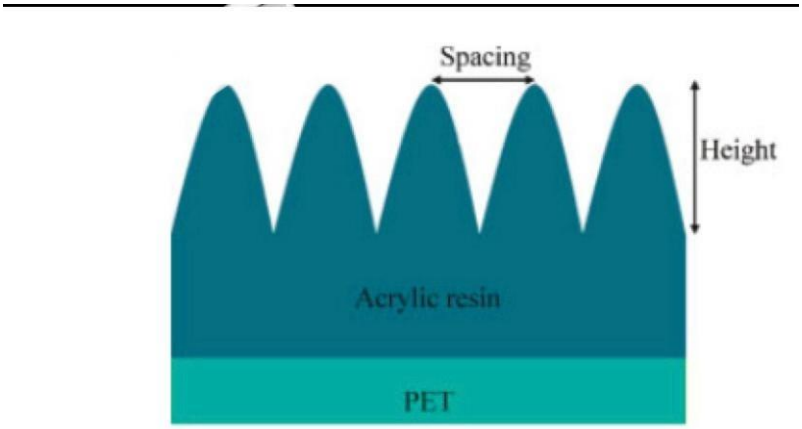


Figure 2: 2D side-view of second RCWA structure

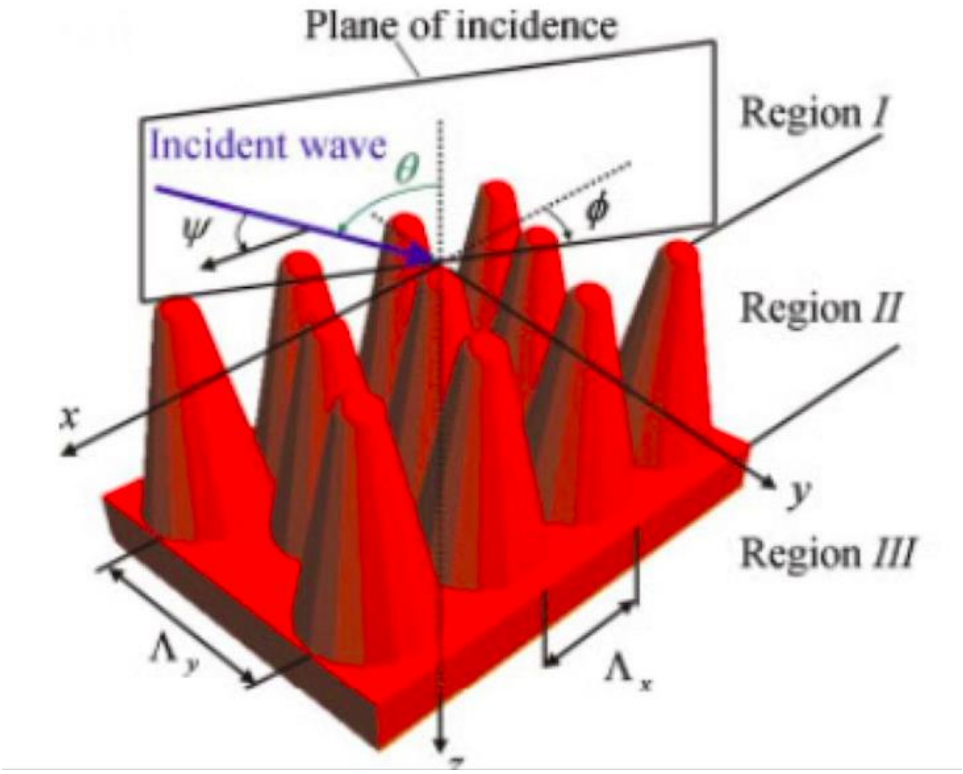


Figure 3: 3D isometric-view of second RCWA structure

Figures 2 and 3 show 2-Dimensional and 3-Dimensional views of the proposed model, respectively.

An RCWA structure consists of two regions, structure and further layers beneath.

The structure rests on top of subsequent layers. The structure being the primary component is of primary importance and lower layers are modified according to increase further accuracy for the RCWA method.

Each component is of a specific material and specific dimensions.

MEEP is designed to operate on the media on the scale of nanometers(nm) only.

MEEP cannot accurately simulate larger dimensions, like millimeters(mm) and centimeters. The specific rationale is explained in chapter 2 MEEP.

Since the corneal array is in the order of nanometers(nm), it can be simulated in MEEP to get valid and precise results. However, if the lower layers cannot be simulated, hence only the corneal structure is simulated.

The 2-Dimensional side view of one nipple is shown in figure 4.

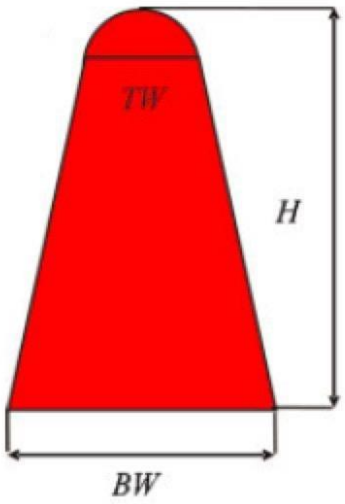


Figure 4: Side-view of a single RCWA nipple

H = 4100–500nm, TW = 450nm, and BW = 490nm

The structure resembles an incomplete conical structure with a smaller hemisphere on the top.

To simulate the structures in FDTD based simulations, software with the ability to calculate multiple computations in the order of nanometers with high accuracy is required. Thus, we use the software MEEP, an FDTD based software.

MEEP is designed to operate on the media on the scale of nanometers(nm) only.

MEEP cannot accurately simulate larger dimensions, like millimeters(mm) and centimeters. The specific rationale is explained further in MEEP chapter 3.

The two proposed structures in Figures 1 and 2 of the nipples are simulated in MEEP, and their respective efficiency is calculated.

The accuracy of each structure is then calculated and illustrated under the range of wavelength -

400 -800 nm.

3 MEEP Software

In this chapter, MEEP software is described, followed by Libraries used in Simulation that are supported by MEEP and other Libraries used to run the Simulations

To implement the proposed models of the Corneal Array, an FDTD (Finite-difference time-domain) software, MEEP is used.

FDTD is a method that is a rigorous and powerful tool for modelling nanoscale optical devices. FDTD solves Maxwell's equations directly without any physical approximation, and the maximum problem size is limited only by the extent of the computing power available.

We use FDTD software as nanoscale computations are necessary for the evaluation of the efficiency of the Structures.

MEEP was originally developed as part of graduate research at MIT. The project has been under continuous development for 20 years. It is currently maintained by an active developer community on GitHub, hence MEEP is open-source and free software under the GNU GPL.

MEEP was developed towards a Vision, ‘To Propel Computational Simulations to the Forefront of Research and Development in Electromagnetics'.

MEEP was initially developed for a C++ interface however, the Python interface is used in this report which came along to be developed after.

MEEP has been used for various domains of problems and research. To execute the evaluation of the Corneal Structure, only certain sections have been discussed that directly or partially pertain to the simulations on the software.

MEEP files use the extension ".ctl" . In the UNIX platform, all the open-source packages are executed in the Terminal.

3.1 Libraries

- **The Math** Library module provides access to the mathematical functions defined by the C standard.
- **Cmath** module provides access to mathematical functions for complex numbers. The functions in the cmath module accept integers, floating-point numbers or complex numbers as arguments. In this report, the cmath library helps in converting complex numbers to respective float integers, to receive a valid result.
- **NumPy** library provides a multidimensional array object for various derived objects' supports Python-sequence input, however it converts such input to NumPy arrays before processing and often outputs NumPy arrays.
- **matplotlib.pyplot** is a collection of functions that make matplotlib work like MATLAB. matplotlib as a whole package is used to make static visualizations of the data, however matplotlib.pyplot is used to make accurate visual illustrations based on the data collected by the end of the simulations
- **meep.materials** is a MEEP specific library, that includes various elements where its physical properties are described by relative permittivity $\varepsilon(\mathbf{x})$ and the relative permeability $\mu(\mathbf{x})$**.** This function is used to import elements into the simulation, instead of accurately specifying their relative permittivity and relative permeability.

Meep simulations are Python scripts that involve specifying the device geometry, materials, current sources, monitor fields, and everything else necessary to set up a calculation. A Python script provides the flexibility to customize the simulation for practically any application.

3.2 Pseudocode

This section first describes all the terminologies that modify a MEEP script and the simulations. The Pseudocode is introduced to throw light on the diction of the code, and each parameter is then described precisely.

Below is the python pseudocode, and each section is further explained.

- Import MEEP and other Libraries
- Set simulation ‘resolution’
- Input ‘source’ parameters
- Define ‘a unit cell’ for simulation (define an area where simulation takes place
- Define the ‘physical properties’ of the corneal structure. // material parameters
- Define the ‘physical dimensions’ of the corneal structure.
- Set PML layer. (Boundary walls of the unit cell)
- Put the final structure of proposed corneal structures, called ‘Geometry
- Set the ‘K’ vector, for the directions of the wave
- Run the simulation and set simulation characteristics
- Capture ‘Transmission’ and ‘reflection’ data from the simulation.
- Calculation of ‘transmission and reflection’
- Plot Illustrative data graphs.

Now we describe how each variable modifies the simulations exactly.

3.2.1 Resolution and Pulse Time

Referencing the pseudocode for resolution. Meep will discretise structures in space and time, and that is specified by a single variable resolution that gives the number of pixels per distance unit. 20 pixels/wavelength in the high-index material. In general, at least 8 pixels/wavelength in the highest dielectric is a promising idea. This will give us a 160×80 cell. Hence as the resolution goes larger computing power increases.

Pulse time is necessary for the light to propagate completely across the cell, in a simulation. Pulse Time shuts off the simulation once the timer runs off, so the reflected light waves do not interfere with the obtained transmission and reflection data.

3.2.2 Source and its parameters

Source is just a source of light. Here we use ‘source’ object from the meep library to define the type of source.

Here we have implicitly defined its parameters. To measure the corneal structure's efficiency accurately, the frequency, wavelength, and range of the frequency are calculated.

However, in the working of a complex moth-eye, the angle of light is vital.

Hence, ' θ’ theta is introduced to specify the angle of incidence and is converted to radians for its calculations.

3.2.3 Unit cell

A cell is a defined area where the simulation takes place, all the simulation and calculations take place within the predefined cell.

A cell can be 2D or 3D (dimensional), since our structure is 3D, we use a 3D cell. For a 3D cell, we use a ‘Vector3’ object from the meep library.

Vector3 stores namely three integer values of x, y and z coordinate on the three-dimensional plane.

3.2.4 PML Layers (perfectly matched layers)

For a light-based simulation to be successful and accurate, ideally there must not be any boundary that may interfere with the simulation.

Hence PML is a fictitious absorbing material with a certain thickness added around the edges of the cell, that does not affect the calculations in the simulation.

In laymen's words, the PML layer is a fictitious layer that ideally absorbs all light, and no light is reflected back to the unit cell.

3.2.5 Physical Parameters

Every physical structure simulated in MEEP may have different physical properties, according to the constraints of the simulation.

Following the common and general physical parameters are defined-

Corneal Structure **(**Proposed)

Structure Parameters(substrate)

A substrate is an underlying layer over which identical corneal structures rest.

i. Lattice Periodicity - Corneal structure is an array of identical repeating structures over a placement. Hence Lattice periodicity defines the interval length of the occurrence of the lattice structures on the placement.
ii. Radius, Height, Thickness - The substrate has specific physical properties like height, radius, thickness,

3.2.6 Geometry

Since two partially different structures are implemented, the two MEEP scripts are distinct on the bases of the Structure.

There are numerous ways to simulate a structure, MEEP allows importing structures from various graphing software. However, we achieve our desired structure by stacking.

Stacking involves putting simple geometric shapes over one another repeatedly to then form a complex whole structure

3.2.7 Material Parameter

In this parameter, the refractive index and conductivity of all physical structures in the simulations are defined.

The refractive index of the substrate and Geometric structures are defined independently and can have the same values.

3.2.8 Structure Geometry

The proposed structure of the corneal array is constructed by further stacking the simplest geometric structures over each other.

The simplest geometric structure MEEP library includes geometric shapes such as spheres, cylinders, and cuboids.

The two distinct corneal structures are constructed by stacking simple geometric objects. A class is created for geometry and objects are called from the MEEP library and are assembled.

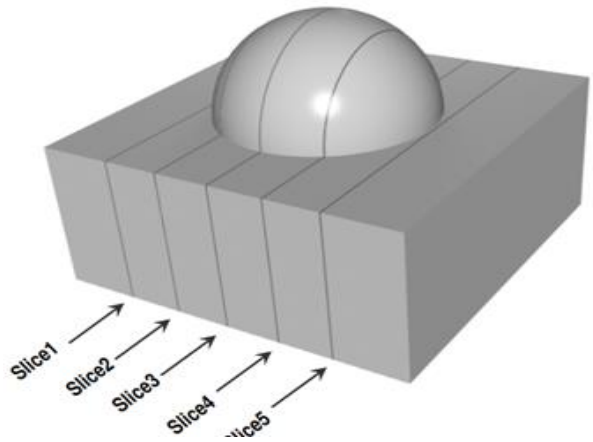


Figure 5: Figure of a complex Arbitrary 3D shape, with simple shapes

3.2.9 'K'- Vector Plane is the vector plane of the wave hitting the structure, since the angle of incidence of light is critical for fair results of the simulation, a vector must be initialised, for all components of the simulation to behave as intended. Plane surface is visualised in Figure 4 chapter 2.3.

3.2.10 Run Simulation(function) -

Simulation is the final object in the MEEP scripts, that calls all previously defined objects, and defines the time of simulation, colour, and for calculation of corneal array, the 'flux' is calculated.

3.2.11 Flux

Flux - Flux is the flow of energy in a medium through a well-defined surface.

Calculation of flux is critical for calculating data, and conclusions.

Flux means 'how much light is transmitted through geometry, and how much is reflected back '.

Computation of reflection and transmittance will precisely exhibit the efficiency of the structure. For an ideal structure, transmittance should be high, and reflection should be close to null.

3.2.12 Illustration

The data collected by the simulation namely ‘transmittance’ and reflectance’ is illustrated by 2-dimensional plots(graphs) for simplicity and to make concise conclusions.

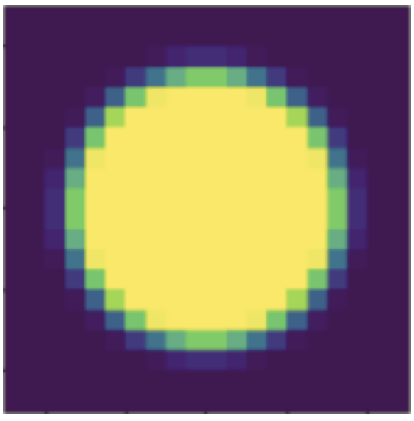
The transmittance and Reflectance data are stored in respective arrays and are plotted on an x-y place via matplotlib.pyplot library.

After understanding the general idea of the pseudocode. The next chapter defines the code and explains the specific code for simulating the two proposed nipple structures.

Before the code, it is essential to visualise the structure of the nipple structure.

Hence here, the structure that is already simulated in MEEP s displayed below.

The working and code of each illustration of the structure is also explained in chapter 3 Code.



The above figures are the side view and top view of the 2nd Structure, respectively.

4 Code

Here we discuss the code for simulation and calculation of the first SWS structure.

In the first part we define basic parameters and variables necessary for simulation, and further explain why and how they are implemented.

The second part describes running the simulation. To run a valid simulation, we perform the simulation twice, one with the nipple structure and one without. Observing the flux patterns with and without the presence of the structure, we normalise our readings and save the data in a text file.

In the final part of the simulation, we see how the collected data is illustrated in a graph.

Towards the end of the chapter, the difference between the first and the second structure is mainly differentiated via a function called ‘geometry.’

“We” in this chapter is referred to as - ‘the writer of the report and the code”.

4.1 First Structure

```
import meep as mp
import math
import cmath
import numpy as np

import matplotlib.pyplot as plt
# from meep.materials import Al

---
```

Import all necessary libraries which are going to be used in the simulation.

Here meep.materials have been commented on because the material of the SWS structure is going to be defined by us.

```
### ------------------------------------------------------------
# Resolution in simulation
resolution = 40     # pixels/um
autoshut = 1E-4
```

Resolution is set to be 40 pixels per micrometer, and the auto-shut timer of the simulation is set at 0.0001 seconds.

```
# Input parameters
# Source parameters
lmin = 0.4      # source min wavelength
lmax = 0.8      # source max wavelength
fmin = 1/lmax       # source min frequency
fmax = 1/lmin       # source max frequency
fcen = 0.5*(fmin+fmax) # center frequency
df = fmax-fmin      # Frequency span
theta_in = 0    # Plane wave incident angle
theta = math.radians(theta_in)  # convert degree to radian
```

Source Parameters:

We define and give values to the min and max wavelength from the data from Structure 1, and theta is only defined.

```
# Structure parameters
a = 0.45          # lattice periodicity
r = 0.1       # nanocylinder radius
h = 0.15            # nanocylinder height
tsub = 4.0          # substrate thickness
tabs = 5.0          # PML thickness
tair = 4.0          # air thickness
sz = tabs+tair+h+tsub+tabs # Total length along the z-axis
```

We give values to the cylinder in the SWS structure, and further define dimensions for PML and air thickness.

The values for thickness were given by trial and error and adjusted towards optimum positions after every trial.

```
# Defined Cell for Calculation
cell_size = mp.Vector3(a,a,sz)  # Unit cell
```

Based on calculations of Structure parameters, we define the unit cell where the simulation takes place.

Vector3 describes a 3-dimensional unit, where values of x and y coordinate is a (lattice periodicity), and we use z as the total thickness of the system.

```
# material parameter
#Sub =  mp.Medium(index=1,D_conductivity=10)
Cyl =  mp.Medium(index=2.5)
Sub = mp.Medium(index=1.5)
```

Here we define the refractive index, of both the main cylinder and the substrate.

For this reason, we did not import the meep.materials library, the refractive index is explicitly defined here instead.

The refractive index of medium = 1, is kept satisfying the natural conditions where the refractive index of air is also 1. However, MEEP always keeps the refractive index of the medium = 1, by default.

The substrate's medium is 1.5, that of glass. Hence the substrate behaves like glass in this simulation.

The cylinder in the simulation need not be given a higher density than the substrate, for effective results in the simulation, thus kept as 2.5.

```
# PML Layer
pml_layers = [mp.PML(thickness=tabs,direction=mp.Z,side=mp.High),
              mp.Absorber(thickness=tabs,direction=mp.Z,side=mp.Low)]
```

Here we define the PML layers. To define specifically we use the ‘z-axis’ scale to refer. Thus, one PML layer is on the top of the z-axis and one on the bottom.

‘High’ and ‘Low’ references to the z-axis can be viewed as the top and the bottom.

The thickness defined is equal to “tabs”, which was defined in chapter 2.2.4 as the thickness of PML layers and has the value of ‘5’.

```
# Structure Geometry
geometry = [ mp.Cylinder(material=Cyl, radius=r, height=h,
                         center=mp.Vector3(0,0,0.5*sz-tabs-tair-0.5*h)),
             mp.Block(material=Sub, size=mp.Vector3(mp.inf,mp.inf,tsub+tabs),
                 center=mp.Vector3(0,0,0.5*sz-tabs-tair-h-0.5*(tsub+tabs))) ]
```

Geometry here is what the structure of the ‘Nipple’(cylinder), and substrate will look like.

We define the cylinder with the material and the dimensions. To define the location on the x, y, and z plane we use the vector3 function and keep the structure in the middle where x = y = 0, and the z point is kept so as not to interfere with PML and other structures.

The substrate on which the geometry rests is defined by giving its refractive index(material). The substrate lies across the x and y plane, and only for a specific thickness in the z-plane.

```
 # k with correct length (plane of incidence: XZ)

k = mp.Vector3(math.sin(theta),0,math.cos(theta))*2*math.pi/(lmax+lmin)/2


src_pos = 0.5*sz-tabs-0.4*tair
sources = [mp.Source(mp.GaussianSource(fcen,fwidth=df),component=mp.Ey,
                     center=mp.Vector3(0,0,src_pos),size=mp.Vector3(a,a,0))]
```

The light from the source should be hitting the geometry on the correct plane. Since the cylinder has its height on the ‘z-axes and radius along the ‘x-y’ plane. The plane of incidence is kept in the ‘x-z’ plane. Now the plane is defined we can define the position and properties of the source that will rest on the ‘x-z’ plane. The Source is used from the MEEP source library, here called ‘mp.source’.

Here the source is only kept as a 2-D object hence has no ‘z-coordinate’. The position is also in the center of the unit cell, directly under which the cylinder rests

```
sim = mp.Simulation(cell_size=cell_size,
#     geometry=geometry,
      sources=sources,
      boundary_layers=pml_layers,
      k_point = k,
      resolution=resolution)
```

To finally define the simulation, all the necessary objects in the simulation are called in the function and redefined.

However, we have not called geometry in the simulation yet. This is done to avoid errors in the simulation. We run ‘sim’ (simulation twice- with and without the geometry). In laymen's terms, we first test the nature flux without the presence of geometry.

In the second simulation, we compare the new flux and the old flux, normalizing it to make better conclusions about the data.

```
nfreq = 4001
 # reflected flux
refl = sim.add_flux(fcen, df, nfreq, mp.FluxRegion(center=mp.Vector3(0,0,0.5*sz-tabs-0.2*tair),size=mp.Vector3(a,a,0)))
tran = sim.add_flux(fcen, df, nfreq, mp.FluxRegion(center=mp.Vector3(0,0,-0.5*sz+tabs+0.2*tsub),size=mp.Vector3(a,a,0)))


sim.run(until_after_sources=mp.stop_when_fields_decayed(25, mp.Ey, mp.Vector3(0,0,0.5*sz-tabs-0.6*tair), autoshut))
```

Reflection and Transmittance, the key functions necessary to calculate the efficiency of the simulation and are defined based on the Fourier mathematical equations.

After defining the simulation ("sim") function, we run the simulation by "sim.run."

We first store the data of reflection and transmission in an array to later compare with the new flux with the geometry. Via this method, we normalise the calculations later which prevents inaccuracy of data.

```
#
sim.reset_meep()
```

We reset the data, to run the simulation with the presence of cylinder and substrate(geometry)

```
#-------------------- Main Calculation
sim = mp.Simulation(cell_size=cell_size,
     geometry=geometry,
     sources=sources,
     boundary_layers=pml_layers,
     k_point = k,
     resolution=resolution)


 # reflected flux
refl = sim.add_flux(fcen, df, nfreq, mp.FluxRegion(center=mp.Vector3(0,0,0.5*sz-tabs-0.2*tair),size=mp.Vector3(a,a,0)))
tran = sim.add_flux(fcen, df, nfreq, mp.FluxRegion(center=mp.Vector3(0,0,-0.5*sz+tabs+0.2*tsub),size=mp.Vector3(a,a,0)))

# for normal run, load negated fields to subtract incident from refl. fields
sim.load_minus_flux_data(refl,Rin_flux)

sim.run(until_after_sources=mp.stop_when_fields_decayed(25, mp.Ey, mp.Vector3(0,0,0.5*sz-tabs-0.6*tair), autoshut))
```

The entire simulation within the unit cell, with geometry, while collecting the flux data.

```
fs = mp.get_flux_freqs(refl)
Rf = mp.get_fluxes(refl)      # Reflection data
Tr = mp.get_fluxes(tran)     # Transmission

R  = np.asarray(Rf)
T  = np.asarray(Tr)
f  = np.asarray(fs)

Ri = np.asarray(Rin)
Ti = np.asarray(Tin)
np.savetxt('test1.txt', (f,Ri,Ti,R,T))
```

The new flux data of reflection and transmission is saved in the new arrays. All the collected data is saved into a '.txt' file, that now will be used to illustrate graphs.

```
R = -R/Ti
T = T/Ti

Lm = 1000/f
plt.figure(10);
plt.plot(Lm,R,'r',label="Reflection")
plt.plot(Lm,T,'b',label="Transmission")
plt.plot(Lm,R+T,'g:',label="Total")
plt.xlabel("Wavelength (nm)")
plt.legend()
plt.show()
```

After normalisation of reflectance and transmittance, we Illustrate the stored data into a graph, with the help of the matplot lib file.

4.2 Second proposed structure.

3.2.1 Geometry

To implement the 2nd Structure, the nipple can be further simplified into two simpler shaped structures. A small semi-sphere resting on a cone, that finally rests on the substrate is implemented in geometry.

From the 4, we add all new dimensions as per the proposed structure.

In materials, the cylinder and Substrate are given new refractive indices, with the substrate having a higher refractive index.

```
# Structure parameters
# Structure parameters
ax = 0.2        # lattice periodicity
#ay = 3**0.5*ax
ay = ax;
Bw = 0.19
Tw = 0.1
h  = 0.55         # moth height
tsub = 4.0          # substrate thickness
tabs = 5.0          # PML thickness
tair = 4.0          # air thickness
sz = tabs+tair+h+tsub+tabs # Total length along the z-axis
```

Updating values from the 2nd structure in chapter 2.3.

```
# Structure
geometry = [ mp.Sphere(material=Cyl, radius=Tw/2,
                       center=mp.Vector3(0,0,0.5*sz-tabs-tair)),
             mp.Cone(material=Cyl, radius=Bw/2, radius2=Tw/2, height=h,
                     center=mp.Vector3(0,0,0.5*sz-tabs-tair-0.5*h)),
                 mp.Block(material=Sub, size=mp.Vector3(mp.inf,mp.inf,tsub+tabs),
                     center=mp.Vector3(0,0,0.5*sz-tabs-tair-h-0.5*(tsub+tabs))) ]
```

Adding a hemisphere on top of the cone in the updated geometry

After updating the two functions the code for the second structure becomes

4.2.2 Code for 2nd Structure

```
import meep as mp
import math
import cmath
import numpy as np

import matplotlib.pyplot as plt
#from meep.materials import Al

### ------------------------------------------------------------------
# Resolution in simulation
resolution = 30     # pixels/um
autoshut = 1E-4

# Input parameters
# Source parameters
lmin = 0.4      # source min wavelength
lmax = 0.8      # source max wavelength
fmin = 1/lmax       # source min frequency
fmax = 1/lmin       # source max frequency
fcen = 0.5*(fmin+fmax) # center frequency
df = fmax-fmin     # Frequency span
theta_in = 0    # Plane wave incident angle
theta = math.radians(theta_in)  # convert degree to radian

# Structure parameters
# Structure parameters
ax = 0.2        # lattice periodicity
#ay = 3**0.5*ax
ay = ax;
Bw = 0.19
Tw = 0.1
h  = 0.55         # moth height
tsub = 4.0          # substrate thickness
tabs = 5.0          # PML thickness
tair = 4.0          # air thickness
sz = tabs+tair+h+tsub+tabs # Total length along the z-axis

# Defined Cell for Calculation
cell_size = mp.Vector3(ay,ax,sz)  # Unit cell


# material parameter
#Sub =  mp.Medium(index=1,D_conductivity=10)
Cyl =  mp.Medium(index=2.5)
Sub = mp.Medium(index=3.5)

# PML Layer
pml_layers = [mp.PML(thickness=tabs,direction=mp.Z,side=mp.High),
              mp.Absorber(thickness=tabs,direction=mp.Z,side=mp.Low)]
```

```
#-------------------- Main Calculation
sim = mp.Simulation(cell_size=cell_size,
      geometry=geometry,
      sources=sources,
      boundary_layers=pml_layers,
      k_point = k,
      resolution=resolution)


 # reflected flux
refl = sim.add_flux(fcen, df, nfreq,
                    mp.FluxRegion(center=mp.Vector3(0,0,0.5*sz-tabs-0.2*tair),
                                  size=mp.Vector3(ay,ax,0)))
tran = sim.add_flux(fcen, df, nfreq,
                    mp.FluxRegion(center=mp.Vector3(0,0,-0.5*sz+tabs+0.2*tsub),
                                  size=mp.Vector3(ay,ax,0)))

# for normal run, load negated fields to subtract incident from refl. fields
sim.load_minus_flux_data(refl,Rin_flux)

sim.run(until_after_sources=mp.stop_when_fields_decayed(25, mp.Ey,
                                                        mp.Vector3(0,0,0.5*sz-tabs-0.6*tair), autoshut))


fs = mp.get_flux_freqs(refl)
Rf = mp.get_fluxes(refl)      # Reflection data
Tr = mp.get_fluxes(tran)     # Transmission

R  = np.asarray(Rf)
T  = np.asarray(Tr)
f  = np.asarray(fs)

Ri = np.asarray(Rin)
Ti = np.asarray(Tin)
np.savetxt('Moth_Out.txt', (f,Ri,Ti,R,T))



R = -R/Ti
T = T/Ti

Lm = 1000/f
plt.figure(10);
plt.plot(Lm,R,'r',label="Reflection")
plt.plot(Lm,T,'b',label="Transmission")
plt.plot(Lm,R+T,'g:',label="Total")
plt.xlabel("Wavelength (nm)")
plt.legend()
plt.show()
```

5 Visualising Structures (Limitation)

Before proceeding to the calculations and conclusions.

It is important to visualise the code only by visualising the structures.

MEEP does not have the ability to visualise structures in real-time; thus, each structure can be improved or changed as per need, but instead -

By running the simulations only to visualise the structure and then make further conclusions. This is one of the limitations of the MEEP software, where a lack of Interface among the layers makes it un-friendly to run simulations.

To get around the issue, the MEEP python code required to visualise structures Is described below.

However, the python code is specifically coded to visualise the proposed Moth eye structures which are coded in the “geometry function” only.

Thus, this python script solves the problem of the lack of presence of an interface in MEEP. Hence this script could be run multiple times to get the required structure of the corneal array.

To visualise and illustrate the geometry, the following MEEP code is the same as the code for simulating a structure. However, instead of calling the ‘sim’ function,

The code necessary for plotting a 3D structure in a 2D plane is used.

The code illustration essentially performs the following tasks:

- Showing the 3D unit cell, into a front view and a top view.
- Generate a refractive-index profile plot. The refractive index plot identifies different refractive indices in the unit cell and calculates its interaction with the light from the source.
- Visualise the data from the refractive index profile plot into easy visually distinguishable colours. Dull colour - Lower refractive index, Bright - Higher refractive index.
- Plot thickness specific planes, that are presented as lines in the 2D plot

5.1 Code to Visualise the Structure.

```
import meep as mp
import math
import cmath

import matplotlib.pyplot as plt
# from meep.materials import Si

### ------------------------------------------------------------
# Resolution in simulation
resolution = 100     # pixels/um

# Input parameters
# Source parameters
lmin = 0.4        # source min wavelength
lmax = 0.8        # source max wavelength
fmin = 1/lmax       # source min frequency
fmax = 1/lmin       # source max frequency
fcen = 0.5*(fmin+fmax) # center frequency
df = fmax-fmin     # Frequency span
theta_in = 0    # Plane wave incident angle
theta = math.radians(theta_in)  # convert degree to radian

# Structure parameters
a = 0.45         # lattice periodicity
r = 0.1       # nanocylinder radius
h = 0.5           # nanocylinder height
tsub = 2.0          # substrate thickness
tabs = 5.0          # PML thickness
tair = 4.0          # air thickness
tlay = 0
sz = tabs+tair+h+tlay+tsub+tabs # Total length along the z-axis

# Defined Cell for Calculation
cell_size = mp.Vector3(a,a,sz)  # Unit cell


# material parameter
Cyl =  mp.Medium(index=2.5)
Sub = mp.Medium(index=3.5)

# PML Layer
pml_layers = [mp.PML(thickness=tabs,direction=mp.Z,side=mp.High),
              mp.Absorber(thickness=tabs,direction=mp.Z,side=mp.Low)]

# Structure Geometry
geometry = [ mp.Cylinder(material=Cyl, radius=r, height=h, center=mp.Vector3(0,0,0.5*sz-tabs-tair-0.5*h)),
               mp.Block(material=Sub, size=mp.Vector3(mp.inf,mp.inf,tsub+tabs),
                            center=mp.Vector3(0,0,0.5*sz-tabs-tair-h-0.5*(tsub+tabs))) ]

## Geometry Plot
sim = mp.Simulation(resolution=resolution,cell_size=cell_size,geometry=geometry)
sim.init_sim()
eps_data = sim.get_epsilon()
```

```
import numpy as np
II = np.asarray(eps_data)
xo,yo,zo = np.shape(II)
print("Geometry shape:",xo,yo,zo)

fig, (ax0,ax1, ax2) = plt.subplots(1, 3)
ax0.imshow(II[:,:,int(zo-(tabs+tair+0.5*h)*resolution)])
ax0.set_title("XY plane in pixel unit")
ax0.set_xlabel("X"); ax0.set_ylabel("Y")

ax1.imshow(np.flipud(np.transpose(II[int(xo/2),:,:])),aspect ='auto')
ax1.set_title("YZ plane at X=0 in pixel unit")
ax1.set_xlabel("Y"); ax1.set_ylabel("Z")

ax2.imshow(np.flipud(np.transpose(II[:,int(yo/2),:])),aspect ='auto')
ax2.set_title("XZ plane at Y=0 in pixel unit")
ax2.set_xlabel("X"); ax2.set_ylabel("Z")
plt.show()


x = np.linspace(-a/2,a/2,xo)
y = np.ones(xo)
fig, (ax0,ax1, ax2) = plt.subplots(1, 3)
ax0.imshow(II[:,:,int(zo-(tabs+tair+0.5*h)*resolution)],extent=[-a/2,a/2,-a/2,a/2])
ax0.set_title("XY plane in pixel unit")
ax0.set_xlabel("X (um)"); ax0.set_ylabel("Y (um)")

ax1.imshow(np.flipud(np.transpose(II[int(xo/2),:,:])),aspect ='auto',
           extent=[-a/2,a/2,-sz/2,sz/2])
ax1.set_title("YZ plane at X=0")
ax1.set_xlabel("Y (um)"); ax1.set_ylabel("Z (um)")

ax2.imshow(np.flipud(np.transpose(II[:,int(yo/2),:])),aspect ='auto',
           extent=[-a/2,a/2,-sz/2,sz/2])
ax2.set_title("XZ plane at Y=0")
ax2.set_xlabel("X (um)"); ax2.set_ylabel("Z (um)")

ax2.plot(x,y*(0.5*sz-tabs-0.5*tair),'r')#,linewidth='3')
ax2.plot(x,y*(-0.5*sz+tabs+0.5*tsub),'b')
plt.show()
```

Hence the result of the above python code to visualise “a geometry’ in MEEP gives the following output via plotting is -

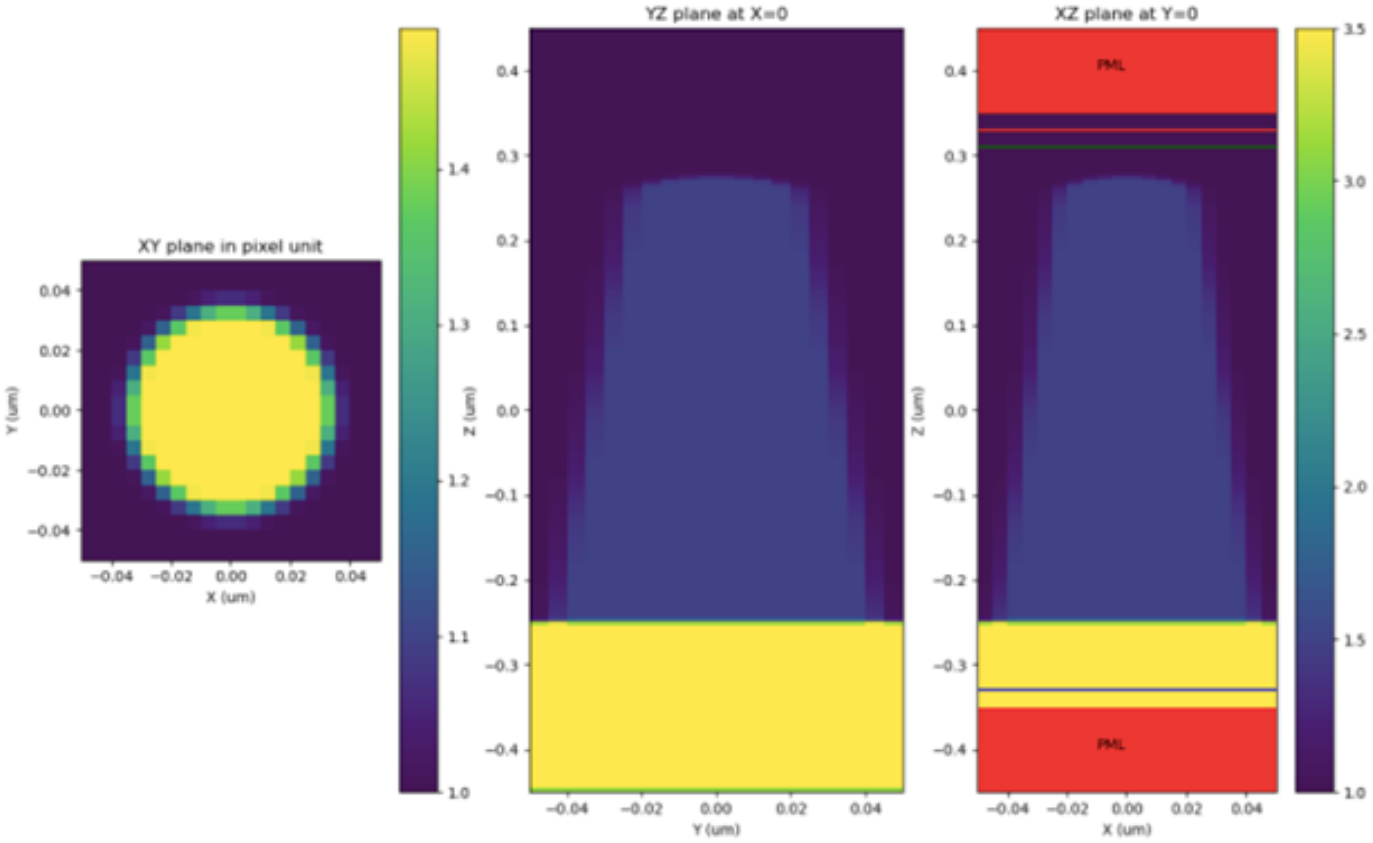


Figure 6: Top and Side Views of the 2nd Structure Simulated via MEEP

Figure 6 is one of the 2D plots, that signify the different planes in the unit cell performing distinct roles. Each line is identifiable by a distinct colour, and the significance of each line is described below.

- Red line: Reflection monitor
- Blue-line: Transmission monitor
- Green-line: Source position
- The ‘Yellow’ layer is the substrate with the highest refractive index.
- Red Layers indicate PML layers.
- Dark Blue is the medium with a refractive index of ‘1’, and light blue stands for the cone of the structure.

6 Results

Results of the Simulation:

Results in this chapter are only analysed and described by the output plots, received after the simulations. The two primary sources of data are

- The graph illustrations are based on the '.txt' file, having data for flux.
- The '.txt' files that hold all the data to make necessary conclusions.

To compare the efficiency of simulated corneal structures, we need two sets of data to compare with.

The first illustration consists of data of the 'reflectance' and 'transmittance', in the absence of structures. This data/graph shows the behavior of reflectance and transmittance of light acting on the substrate only

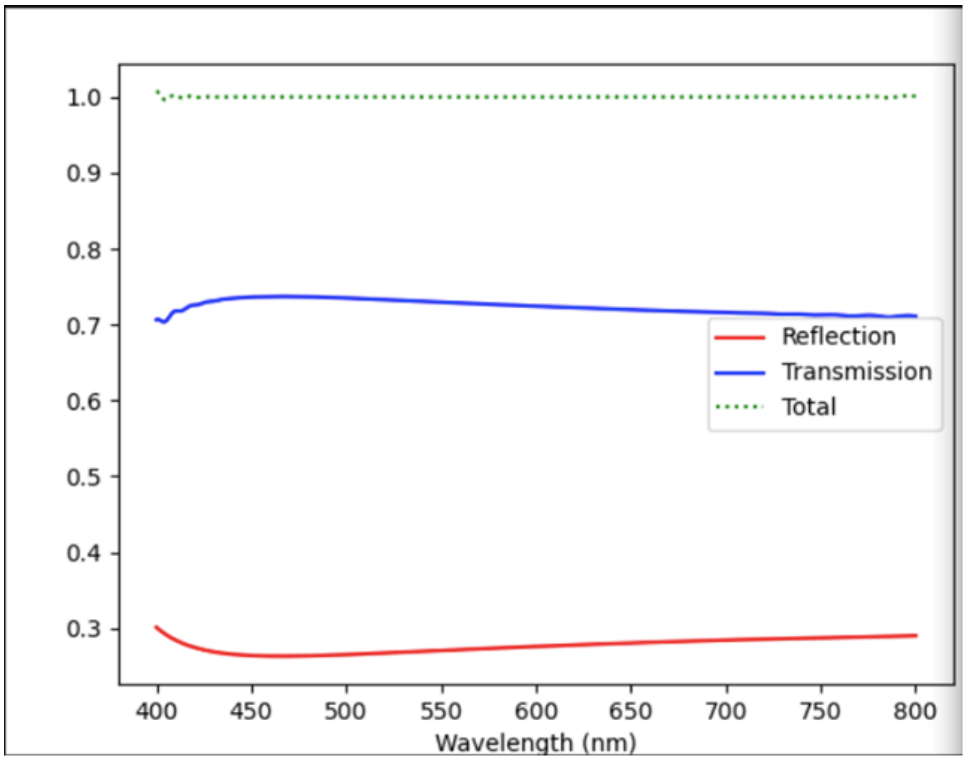


Figure 7: Flux Pattern in the presence of Substrate only

In figure 7, three distinct colour lines signify the parameter the illustration displays in this illustration the red line shows the ratio of light reflected and the blue similarly shows the amount of light transmitted. However, the total amount of light = light reflected + light transmitted.

Hence the total will always be 1 or 100%.

As in the figure 7 illustration, In the presence of the substrate alone, the reflection and transmission vary in behavior over the spectrum of the wavelength. In other words, the behavior is dependent on the wavelengths from the source.

Analysing data from the '.txt' file, the average transmission rate of the substrate is 70.73% and the average reflection is 29.27%.

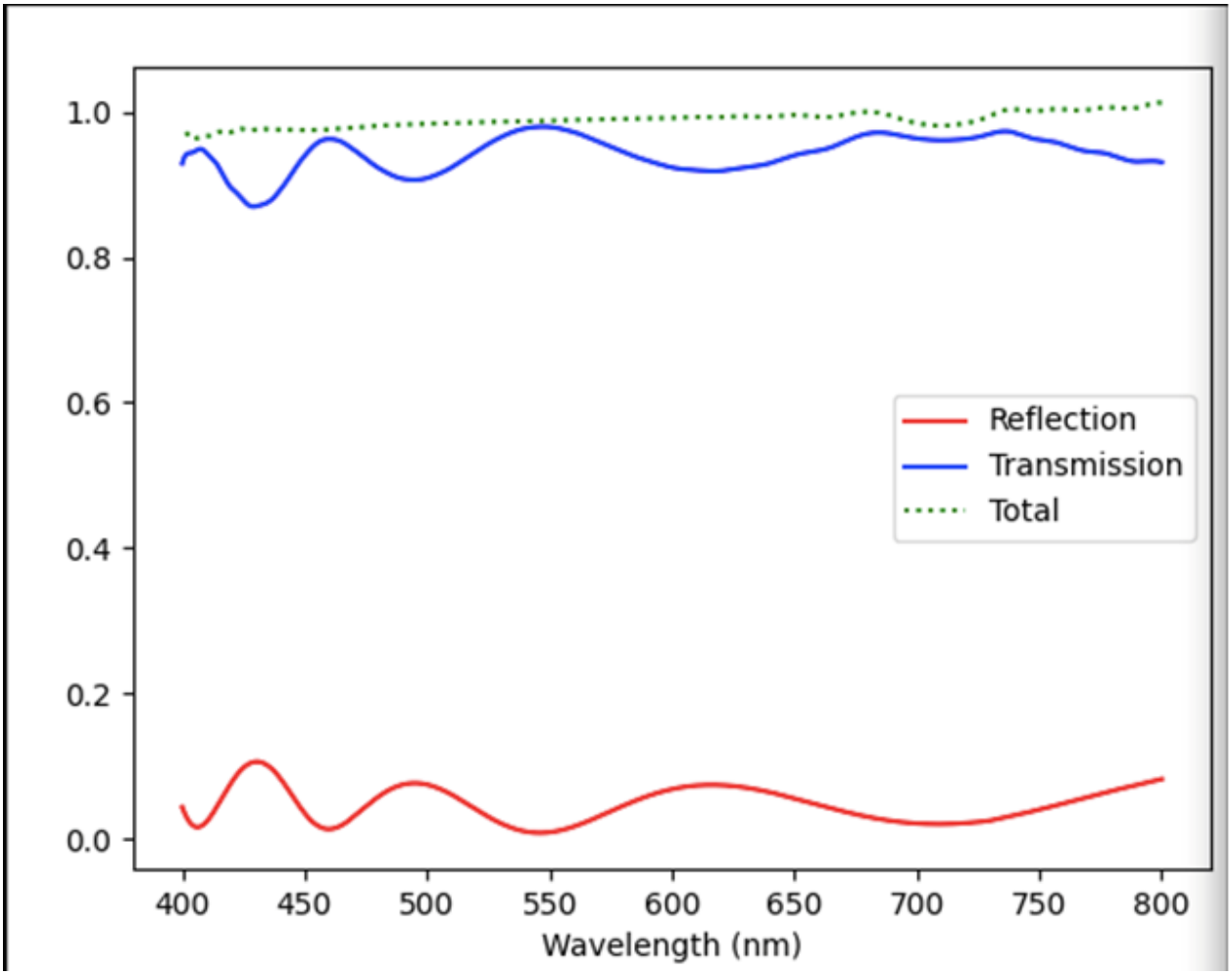


Figure 8: Flux Pattern in the presence of Substrate only

In Figure 8, it is clear and visible, displaying the efficiency of the second simulated 'Nipple' Structure. The uniform, continuous nature of the functions of reflection and transmission conveys the data is not incorrect and can be used to make conclusions.

The Total function is always close to =1; the total value of 'Total' cannot be unity due to the minor interferences of light waves in the simulation. However, the slight errors are due to the process of reading the data. Because MEEP is FDTD based software there is almost zero error in the accuracy of the simulations.

The average Transmission between wavelengths of 400 and 800 is 91.34%, and reflection is 8.66%.

However, at particular wavelengths near 450nm, 550nm and 700nm, the transmission is up to 99.28% and less than 1% reflectance in some wavelengths, according to the data of flux from the '.txt file'.

7 Conclusions

With the data of flux from the simulations, the presence of the proposed 'nipple array' pattern on a substrate, makes it 28.57% more efficient than just a substrate.

Hence the presence of a high refractive index in either substrate or the nipple, the efficiency of the proposed optimised nipple structure will be 28.57% more efficient in transmitting energy through itself, as compared to the efficiency of a surface without the 'nipple structure'.

Hence in the case of solar panels, the substrate can be a supporting base, to hold the structure and provide rigidity and protection from the environment, and the layer of repeating 'corneal structures will show increased efficiency in generating electricity. However, the increased transmission rate of 28.57% is not directly proportional to the efficiency of energy generation.

However, the presence of a corneal layer will increase efficiency and reduce energy loss due to reflection by the surface from a solar cell.

7.1 Future Work

MEEP software can only execute correct and accurate simulations when the Input variables and data are calculated and exact.

It is exceedingly difficult to identify and debug errors in the MEEP script, mainly due to the large amount of compile-time for solving multiple Fourier Transforms and other calculations in a simulation.

However, the presence of an Interface in which, making a structure could be simplified either via selecting preexisting complex shapes from a library or a visual system such as Mayavi or Matlab could speed up the process from the end of the user. For now, MEEP only runs on the terminal, the presence of an executable file with built-in GitHub control will also reduce a lot of time while updating the repository being used. Several interface layers have the possibility to be built over the present MEEP software, to simplify and make simulations friendly, with lesser use of python script functions.

However, MEEP is the most accurate software yet for accurate FDTD calculation-based simulations.

The optimised corneal structure can be confidently added over the glass surfaces. However, a sturdy and resistant material is required for large scale adoption.

If in future a similar economic and effective material with a different refractive index is either found or manufactured, it can be first simulated in MEEP in the similar way, with its new refractive index, to test its effectiveness and efficiency, and can be further altered as per needs, via the MEEP software.

Bibliography

- Haidong Mou, Weihong Hua, and Zining Yang, "Simulation of diffraction effects of anti-reflection microstructures used as intra-cavity optical elements," Opt. Express 24, 5069-5077 (2016)
- Yamada, N., Kim, O.N., Tokimitsu, T., Nakai, Y. and Masuda, H. (2011), Optimization of anti-reflection moth-eye structures for use in crystalline silicon solar cells. Prog. Photovolt: Res. Appl., 19: 134-140.
- Vazquez F. Welcome to the New Journal *Biomimetics*. Biomimetics (Basel). 2016 Feb 25;1(1):1. doi: 10.3390/biomimetics1010001. PMCID: PMC6477595.
- Chapman, R., Simpson, S. and Douglas, A., 2012. *The Insects*. Cambridge: Cambridge University Press, p.Ch.22.

- Oskooi, Ardavan F. et al. “Meep: A flexible free-software package for electromagnetic simulations by the FDTD method.” Computer Physics Communications 181.3 (2010): 687-702. Version: Author's final manuscript
- C. Rumpf Raymond. “Lecture 17 (FDTD) -- Power flow and PML placement” YouTube Video, 59.09. Apr 22, 2014. https://www.youtube.com/watch?v=IsFuJVjruqk