Universidade Federal
de Campina Grande

Trabalho final
Cruzamento com baixa perda

Alunos:
ANA JULIA FERNANDES DE BRITO AMENO - 123211003

Professor:
adolfofh@dee.ufcg.edu.br

Introdução

Este relatório trata sobre um projeto de cruzamento entre 2 guias fotônicos com baixa perda e baixa interferência, ele é baseado no artigo de Bogaerts (1).
É feito sweeps de profundidade de etch e de diferentes comprimentos de onda para conferir a banda de propagação como também qual profundidade proporciona menor perda e interferência.

1. Inicialização do software com API de Python

A primeira parte do código é dedicada à inicialização do ambiente de simulação. Para isso, utilizamos a API do Lumerical chamada lumapi, que permite a integração do Lumerical com Python.
A inicialização consiste na importação das bibliotecas que serão usadas, na chamada da api lumapi como também em iniciar o ambiente MODE do Lumerical.

```
import sys, os
import importlib.util
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import lambda2nu, nu2lambda, c
```

```
from IPython.core.display import HTML
from IPython.core.pylabtools import figsize
from IPython.display import display, Math
```


```
from IPython.core.display import HTML
from IPython.core.pylabtools import figsize
from IPython.display import display, Math
```


```
lumapiFile = r"C:\\Program Files\\Lumerical\\v221\\api\\python\\lumapi.py"
spec = importlib.util.spec_from_file_location("lumapi", lumapiFile)
lumapi = importlib.util.module_from_spec(spec)
spec.loader.exec_module(lumapi)
modeapi = lumapi.MODE(hide = False)
```

2. Definição das constantes e materiais

Nesta etapa definimos as grandezas de micrômetro e nanômetro para facilitar a visualização dos valores no código.

```
um = 1e-6
nm = 1e-9
```

Definimos também os materiais usados na simulação que nesse caso serão o Sio2 e o Si, nesta simulação também utilizamos o ar, porém não é necessário definir ele pois ele é o material padrão do Lumerical quando não há nenhum material selecionado.

```
# Materials
material_Clad = "SiO2 (Glass) - Palik"
material_Si = "Si (Silicon) - Dispersive & Lossless"

# Add material Si
matname = material_Si
matReturn = modeapi.getmaterial(material_Si)
if (modeapi.findstring(matReturn, 'is not available') != -1):
    newmaterial = modeapi.addmaterial("Lorentz")
    modeapi.setmaterial(newmaterial, "name", matname)
    modeapi.setmaterial(matname, "Permittivity", 7.98737492)
    modeapi.setmaterial(matname, "Lorentz Linewidth", 1e8)
    modeapi.setmaterial(matname, "Lorentz Resonance", 3.93282466e+15)
    modeapi.setmaterial(matname, "Lorentz Permittivity", 3.68799143)
    modeapi.setmaterial(matname, "color", np.array([0.85, 0, 0, 1]))  # red
```

```
# Add material Clad
matname = material_Clad
matReturn = modeapi.getmaterial(material_Clad)
if (modeapi.findstring(matReturn, 'is not available') != -1):
    newmaterial = modeapi.addmaterial("Lorentz")
    modeapi.setmaterial(newmaterial, "name", matname)
    modeapi.setmaterial(matname, "Permittivity", 2.119881)
    modeapi.setmaterial(matname, "Lorentz Linewidth", 1e10)
    modeapi.setmaterial(matname, "Lorentz Resonance", 3.30923843e+15)
    modeapi.setmaterial(matname, "Lorentz Permittivity", 49.43721)
    modeapi.setmaterial(matname, "color", np.array([0.5, 0.5, 0.5, 1]))  # grey
```

3.Criação da estrutura do guia de onda

Após configurar a inicialização, partimos para a criação da estrutura.

Primeiramente, alternamos para o modo de layout no Lumerical para permitir a edição e limpar qualquer objeto que já exista.

Em seguida, criamos um grupo de estruturas onde será inserida toda geometria do projeto, deixando-o mais organizado.

```
modeapi.switchtolayout()

modeapi.deleteall()
```

```
modeapi.addstructuregroup()
dev_layer = 'taper_cross'
modeapi.set('name',dev_layer)
modeapi.set("x", 0)
modeapi.set("y", 0)
modeapi.set("z", 0)
```

É necessário definir os parâmetros dos guias antes de criá-los, e é isso que foi feito abaixo

```
#wavelenght
lambda_0 = 1550*nm

# Waveguide parameters
wvg_width = 500 * nm
wvg_height = 220 * nm
wvg_lenght = 9 * um

# Crossing parameters
cross_lenght = 6 * um
shlwetch_width = 800 * nm #the shallow etched final width when it gets to the
crossing
curve_width = 2.5 * um
shlwetch_depth = 50 * nm
deepetch_depth = 220 * nm
```

Nessa etapa cria-se o cladding, ou seja, a base onde o circuito ficará.

```
modeapi.select("Glass_palik")
modeapi.delete()

modeapi.addrect()
modeapi.set("name", "Glass_palik")
modeapi.set("x", 0)
modeapi.set("x span", 12 * um)
modeapi.set("y", 0)
modeapi.set("y span", 12 * um)
modeapi.set("z", -1*um)
modeapi.set("z span",2 * um)
modeapi.set("material", material_Clad)

# Middle square
modeapi.select(f"{dev_layer}::box")
modeapi.delete()
modeapi.addrect()
modeapi.addtogroup(dev_layer)
modeapi.set("name", "box")
modeapi.set("x", 0)
modeapi.set("x span", shlwetch_width)
modeapi.set("y", 0)
modeapi.set("y span", shlwetch_width)
modeapi.set("z", (wvg_height) / 2)
```

```
modeapi.set("z span",(wvg_height))
modeapi.set("material", material_Si)
```

Em seguida criamos os tapers e guias de entrada e saída, essa etapa é toda feita num loop para evitar repetição de código.

```
# Tapers
for i in range(4):

    #making the taper port
    modeapi.select(f"{dev_layer}::taper port {str(i + 1)}")
    modeapi.delete()
    modeapi.addobject("linear_taper")
    modeapi.addtogroup(dev_layer)
    modeapi.set("name", f"taper port {str(i + 1)}")
    modeapi.set("thickness", wvg_height)
    modeapi.set("angle_side", 90)
    modeapi.set("width_l", shlwetch_width)
    modeapi.set("width_r", wvg_width)
    modeapi.set("len", (cross_lenght-(800*nm))/2)
    modeapi.set("material", material_Si)
    modeapi.set("z", (wvg_height) / 2)

    #making the waveguides
    modeapi.select(f"{dev_layer}::waveguide {str(i + 1)}")
    modeapi.delete()
    modeapi.addrect()
    modeapi.addtogroup(dev_layer)
    modeapi.set("name", f"waveguide {str(i + 1)}")
    modeapi.set("z span", wvg_height)
    modeapi.set("material", material_Si)
    modeapi.set("z", wvg_height / 2)
    yspan = wvg_width
    xspan = 3 * um

    if(i<2):

        #making the curved tapers
        modeapi.select(f"{dev_layer}::curved taper {str(i + 1)}")
        modeapi.delete()
        modeapi.addcustom()
        modeapi.addtogroup(dev_layer)
        modeapi.set("name", f"curved taper {str(i + 1)}")
        modeapi.set("material", material_Si)
        modeapi.set("z", (wvg_height/2)-shlwetch_depth)
        modeapi.set("y", modeapi.getnamed(f"{dev_layer}::box", "y"))
        modeapi.set("x", modeapi.getnamed(f"{dev_layer}::box", "x"))
        modeapi.set("z span", wvg_height)
        modeapi.set("y span", cross_lenght/2)
        modeapi.set("x span", cross_lenght)
```

```
        modeapi.set("create 3D object by","extrusion")
        modeapi.set("equation 1", "-0.1111111111111111*x^2+1.25")

        #defines the position of the taper port
        x0 =  modeapi.getnamed(f"{dev_layer}::box", "x max")+
modeapi.getnamed(f"{dev_layer}::taper port {str(i + 1)}", "len")/2
        y0 = modeapi.getnamed(f"{dev_layer}::box", "y")

        #defines the position of the waveguides
        x1 = modeapi.getnamed(f"{dev_layer}::box", "x") + (cross_lenght+ 2 * um)/2
        y1 = modeapi.getnamed(f"{dev_layer}::box", "y")

        #defines the direction of the ports(1 and 2 are horizontal and 3 and 4
vertical)
        rot_1=0

    else:

        #defines the position of the taper port
        y0 =  modeapi.getnamed(f"{dev_layer}::box", "y max")+
modeapi.getnamed(f"{dev_layer}::taper port {str(i + 1)}", "len")/2
        x0 = modeapi.getnamed(f"{dev_layer}::box", "x")

        #defines the position of the waveguides
        x1 = modeapi.getnamed(f"{dev_layer}::box", "x")
        y1 = modeapi.getnamed(f"{dev_layer}::box", "y") + (cross_lenght+ 2 * um)/2

        #defines the direction of the ports(1 and 2 are horizontal and 3 and 4
vertical)
        rot_1=90

    if((i + 1) % 2 == 0):

        #fixes the positions of the ports so they are not only locates in the
positive
        x0 = -x0
        x1 = -x1
        #selects the right side to turn the tapers
        scd_axis = "y"

    else:
        #fixes the positions of the ports so they are not only locates in the
positive
        y0 = -y0
        y1 = -y1
        #selects the right side to turn the tapers
        scd_axis = "x"

    #sets the values defined prior to the tapers
```

```
    modeapi.select(f"{dev_layer}::taper port {str(i + 1)}")
    modeapi.set("x", x0)
    modeapi.set("y", y0)
    modeapi.set("first axis", "z")
    modeapi.set("rotation 1", rot_1)
    modeapi.set("second axis", scd_axis)
    modeapi.set("rotation 2", 180)

    #sets the values defined prior to the waveguides
    modeapi.select(f"{dev_layer}::waveguide {str(i + 1)}")
    modeapi.set("x", x1)
    modeapi.set("y", y1)
    modeapi.set("y span", yspan)
    modeapi.set("x span", xspan)
    modeapi.set("first axis", "z")
    modeapi.set("rotation 1", rot_1)

#turns one of the curved tapers vertically
modeapi.select(f"{dev_layer}::curved taper 2")
modeapi.set("first axis", "z")
modeapi.set("rotation 1", 90)
```

4. Configuração da Simulação VarFDTD

Para analisar o objeto que criamos necessitamos adicionar e configurar o simulador, que nesse caso é o VarFDTD.

```
#Adding varFDTD
modeapi.select('varFDTD')
modeapi.delete()
modeapi.addvarfdtd()
modeapi.set('Mesh accuracy', 4)
modeapi.set("x", 0)
modeapi.set("x span", 10 * um)
modeapi.set("y", 0)
modeapi.set("y span", 10 * um)
modeapi.set("z", 0.1*um)
modeapi.set("z span",1.2 * um)

modeapi.set("x0", -4.5*um)


# Starting test_points with a numpy matrix
test_points = np.zeros((2, 4))

test_points[0, 0]= -3.4 * um
test_points[0, 1]= 0 * um
test_points[0, 2]= 0.5* um
test_points[0, 3]= 2* um
```

```
test_points[1, 0]= 0* um
test_points[1, 1]= 2.5* um
test_points[1, 2]= -2.5* um
test_points[1, 3]= 0* um

modeapi.set("test points",test_points)
```

```
#setting the wavelenght variables
modeapi.setglobalmonitor("frequency points",20)
modeapi.setglobalmonitor("use source limits",0)
modeapi.setglobalmonitor('minimum wavelength', wvlngth_start)
modeapi.setglobalmonitor('maximum wavelength', wvlngth_stop)
modeapi.setglobalsource('wavelength start', wvlngth_start)
modeapi.setglobalsource('wavelength stop', wvlngth_stop)
```

O VarFDTD se faz necessário adicionar uma fonte.

```
modeapi.select("source")
modeapi.delete()
modeapi.addmodesource()
modeapi.set("x", -4.9*um)
modeapi.set("y", 0)
modeapi.set("y span",(wvg_width+1)*um )
```

Para as imagens ficarem mais claras colocamos um mesh e junto dele os monitores para podermos extrair os dados após rodar a simulação.

```
modeapi.select("mesh")
modeapi.delete()
modeapi.addmesh()
modeapi.set("x", 0)
modeapi.set("x span", 10 * um)
modeapi.set("y", 0)
modeapi.set("y span", 10 * um)
modeapi.set("z", 0.11*um)
modeapi.set("z span",0.22 * um)



modeapi.select('index')
modeapi.delete()
modeapi.addindex()
modeapi.set("name", "index")
modeapi.set('monitor type',"2D Z-normal")
modeapi.set("x", 0)
modeapi.set("x span", 10 * um)
modeapi.set("y", 0)
modeapi.set("y span", 10 * um)
modeapi.set("z", wvg_height/2)

modeapi.select('monitor')
modeapi.delete()
modeapi.addprofile()
```

```
modeapi.set('monitor type',"2D Z-normal")
modeapi.set("x", 0)
modeapi.set("x span", 10 * um)
modeapi.set("y", 0)
modeapi.set("y span", 10 * um)
modeapi.set("z", wvg_height/2)

modeapi.select('in_1')
modeapi.delete()
modeapi.addpower()
modeapi.set("name", "in_1")
modeapi.set("monitor type", 5)
modeapi.set("x", -4.5*um)
modeapi.set("y", 0)
modeapi.set("y span",(wvg_width+1)*um )
modeapi.set('z',wvg_height/2)
modeapi.set('z span', (wvg_height+1)*um )

modeapi.select('out_1')
modeapi.delete()
modeapi.addpower()
modeapi.set("name", "out_1")
modeapi.set("monitor type", 5)
modeapi.set("x", 4*um)
modeapi.set("y", 0)
modeapi.set("y span",(wvg_width+1)*um )
modeapi.set('z',wvg_height/2)
modeapi.set('z span', (wvg_height+1)*um )
```

4. Configuração do sweep

Para testarmos qual o melhor resultado é necessário um sweep da simulação, neste caso é feito um sweep da profundidade do etch entre 80 nm e 30 nm, e pegamos os resultados do campo elétrico e da entrada e saída, na entrada e saída pegamos especificamente o resultado do campo da transmitância e da potência.

```
# Adding sweep
modeapi.switchtolayout()
sweep_name = 'test_one'
modeapi.deletesweep(sweep_name);
modeapi.addsweep()
modeapi.setsweep('sweep','name',sweep_name)
modeapi.setsweep(sweep_name, "type", "Ranges")
modeapi.setsweep(sweep_name, "number of points", 9)


# Setting up the sweep parameters, and choosing the range of the sweep
para = {}
para['Name']= "depth curved taper 1"
para['Parameter']= '::model::taper_cross::curved taper 1::z'
```

```
para['Start']= ((wvg_height/2)-shlwetch_depth)+0.03*um
para['Stop']= -(((wvg_height/2)-shlwetch_depth)+0.03*um)
modeapi.addsweepparameter(sweep_name,para)

para = {}
para['Name']= "depth curved taper 2"
para['Parameter']= '::model::taper_cross::curved taper 2::z'
para['Start']= ((wvg_height/2)-shlwetch_depth)+0.02*um
para['Stop']= -(((wvg_height/2)-shlwetch_depth)+0.02*um)
modeapi.addsweepparameter(sweep_name,para)

resu = {}
resu['Name']= "field monitor"
resu['Result']= '::model::monitor::E'
modeapi.addsweepresult(sweep_name,resu)

resu = {}
resu['Name']= "in monitor"
resu['Result']= '::model::in_1::E'
modeapi.addsweepresult(sweep_name,resu)

resu = {}
resu['Name']= "out monitor"
resu['Result']= '::model::out_1::E'
modeapi.addsweepresult(sweep_name,resu)

resu = {}
resu['Name']= "in monitor"
resu['Result']= '::model::in_1::P'
modeapi.addsweepresult(sweep_name,resu)

resu = {}
resu['Name']= "out monitor"
resu['Result']= '::model::out_1::P'
modeapi.addsweepresult(sweep_name,resu)

resu = {}
resu['Name']= "in monitor"
resu['Result']= '::model::in_1::T'
modeapi.addsweepresult(sweep_name,resu)

resu = {}
resu['Name']= "out monitor"
resu['Result']= '::model::out_1::T'
modeapi.addsweepresult(sweep_name,resu)
```

Assim rodamos a simulação, o sweep e salvamos o arquivo.

```
modeapi.run()
```

```
modeapi.runsweep(sweep_name)
```

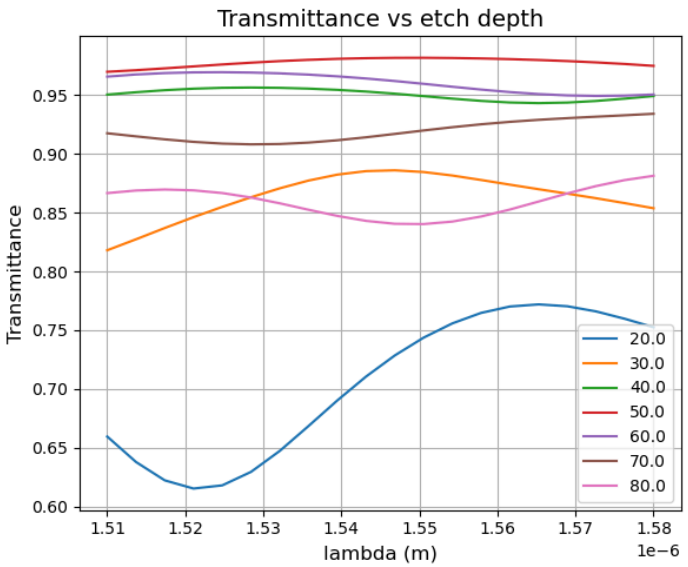
```
modeapi.save(filename_var)
```

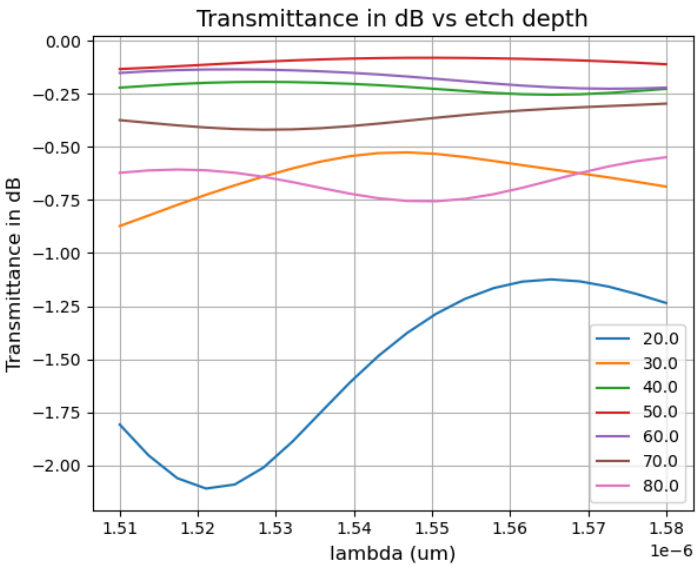
6.Resultados

Os primeiros resultados extraídos são da transmitância, nele podemos ver claramente qual profundidade do etch tem maior e menor perda através de um gráfico linear.

O código a seguir faz a extração do resultado do sweep e em seguida plota os gráficos.

```
 #getting the transmittance result
 T_data = np.abs(modeapi.getsweepdata(sweep_name, 'out transmittance'))
 np.shape(T_data)
 # Create x-axis values (assuming equally spaced points)
 x = np.linspace(1.58e-6, 1.51e-6, T_data.shape[0])
 etch_depth= (np.linspace((shlwetch_depth-sweep_range)/nm,
(shlwetch_depth+sweep_range)/nm, 7))

 fig, axs = plt.subplots(1,2,  figsize=(20, 10))
 for i in range(7):
     axs[0].plot(x,  T_data[:, (i*int((T_data.shape[1]-1)/6))], label=f'Re(T)
Indices: {T_data[:,  (i*int((T_data.shape[1]-1)/6))] }')
     axs[1].plot(x, 10*np.log10(T_data[:, (i*int((T_data.shape[1]-1)/6))]),
label=f'Re(T) Indices: { (i*int((T_data.shape[1]-1)/6)) + 1}')
 # Graph 1: Transmittance
 axs[0].set_xlabel('lambda (m)',fontsize=12)
 axs[0].set_ylabel('Transmittance', fontsize=12)
 axs[0].set_title('Transmittance vs etch depth', fontsize=14)
 axs[0].grid()
 axs[0].legend(etch_depth)
 # Graph 1: Transmittance in dB
 axs[1].set_xlabel('lambda (um)', fontsize=12)
 axs[1].set_ylabel('Transmittance in dB', fontsize=12)
 axs[1].set_title('Transmittance in dB vs etch depth', fontsize=14)
 axs[1].grid()
 axs[1].legend(etch_depth)

 plt.tight_layout()
 plt.grid(True)
```





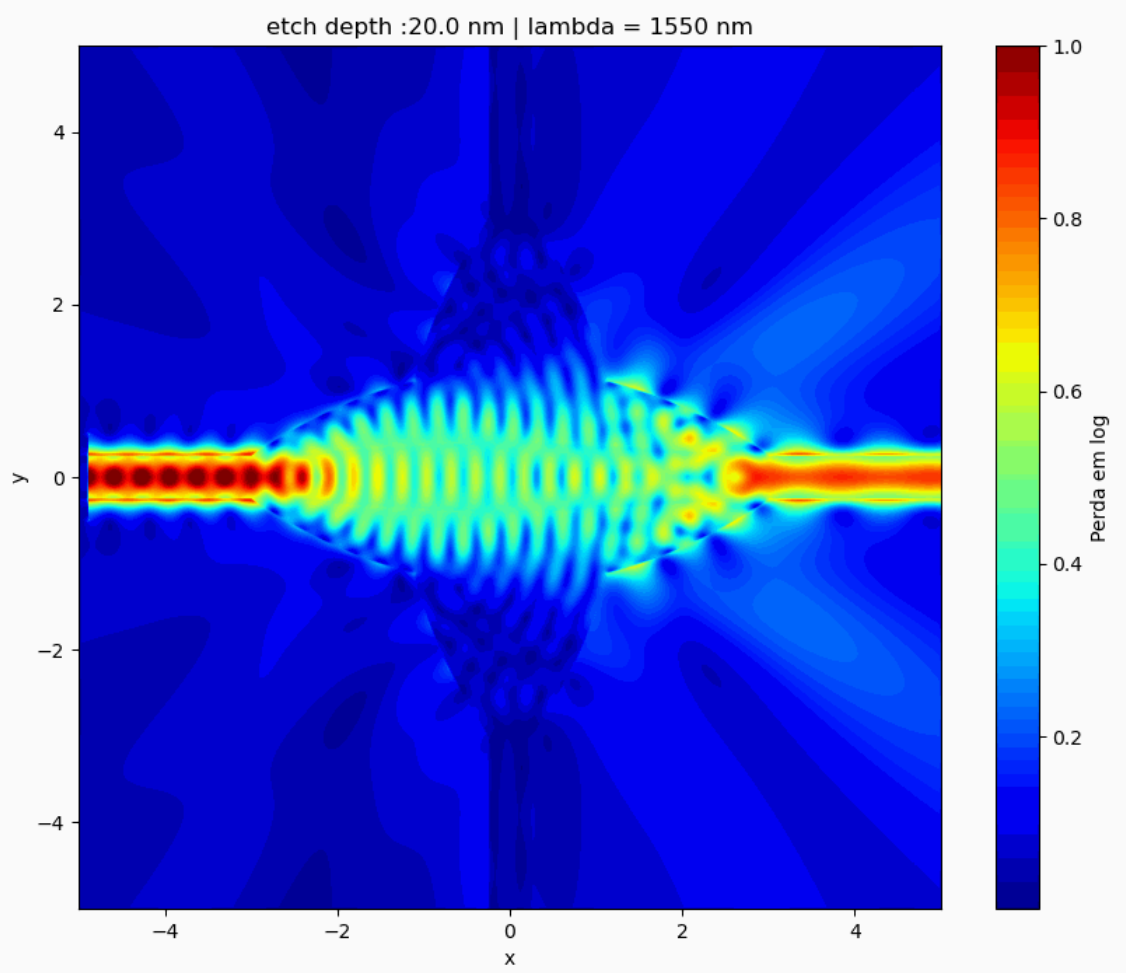
Observando os resultados vemos claramente que a profundidade em torno de 50nm é a que melhor se propaga na banda C.

Para confirmar esse resultado e ter gráficos que mostrem essa perda mais claramente, plotamos agora o campo elétrico clássico e em log.

```
field_data = (modeapi.getsweepdata(sweep_name, 'field monitor'))
reshaped_data = np.abs(np.reshape(field_data, (801, 801, 3, 20, nop)))
heatmap_mag = np.sqrt((reshaped_data[:,:, 0, :,:])**2 + (reshaped_data[:,:, 1,
:,:])**2 + reshaped_data[:,:, 2, :,:]**2)
```

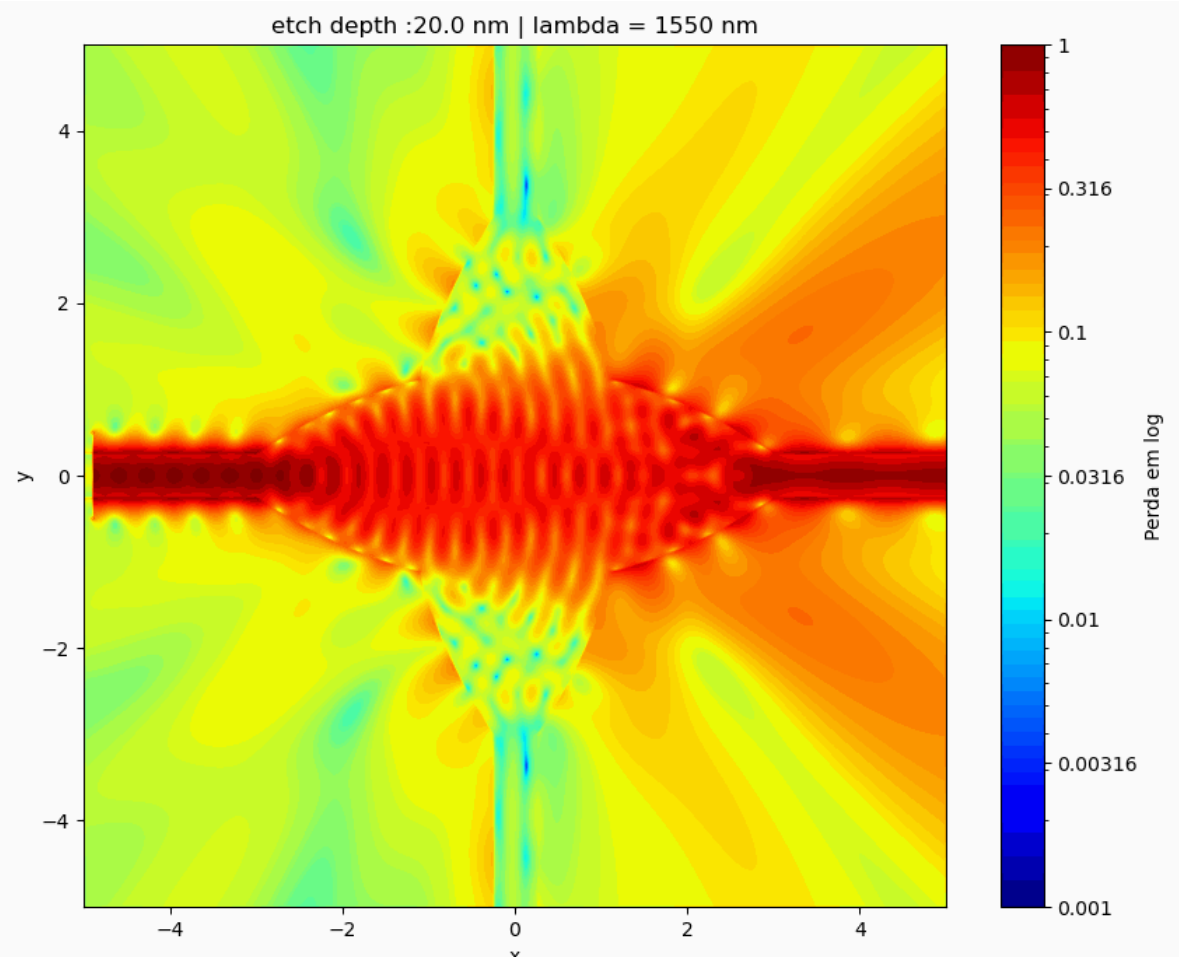
```
# Define path to save images
save_path = 'C:/Users/juame/Desktop/'
os.makedirs(save_path, exist_ok=True)
etch_depths = np.round(np.linspace((shlwetch_depth-sweep_range)/nm,
(shlwetch_depth+sweep_range)/nm, nop))

# List of frames
frames = []

# generating graphs and saving as frames
for i in range(nop):
    plt.figure(figsize=(10, 8))
    plt.imshow(heatmap_mag[:, :, 9, i], cmap='jet',vmin=0.001, vmax=1,extent=[-5,
5, -5, 5])

    # Adding colorbar
    plt.colorbar(label='Perda em log')

    # Adding titles and labels
    plt.title(f'etch depth :{etch_depths[i]} nm | lambda = 1550 nm')
    plt.xlabel('x')
    plt.ylabel('y')

    # Save images
    file_path = os.path.join(save_path, f'frame_{i}.png')
    plt.savefig(file_path)
    plt.close()
```

```
    # Load GIF as frame
    frame = Image.open(file_path)
    frames.append(frame)

# Create GIF and save it on the PC
gif_path = os.path.join(save_path, 'animation.gif')
frames[0].save(gif_path, save_all=True, append_images=frames[1:], optimize=True,
duration=100, loop=0)

# Showing gif in code
display(IPImage(filename=gif_path))

# showing path where GIF was saved in the pc
print(f"GIF saved in: {gif_path}")
```

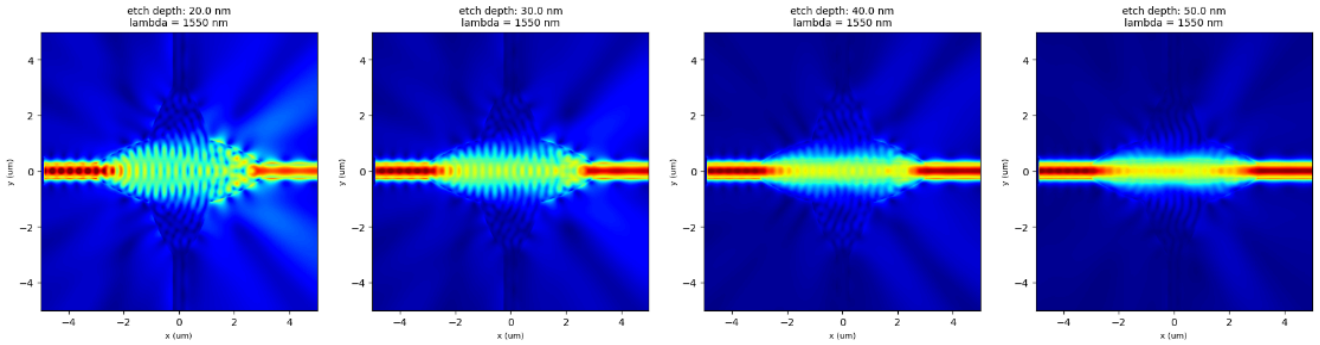


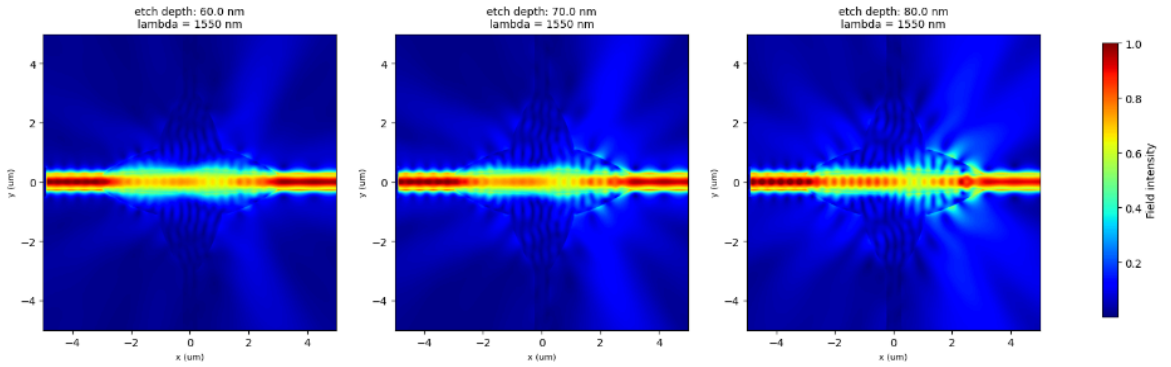

```
# Defines path to save images
save_path = 'C:/Users/juame/Desktop/'
os.makedirs(save_path, exist_ok=True)


# List of frames
frames = []

# Generate and save frames
for i in range(int(nop)):
```

```
    plt.figure(figsize=(10, 8))
    plt.imshow(heatmap_mag[:, :, 9, i], cmap='jet',
norm=colors.LogNorm(vmin=0.001, vmax=1),extent=[-5, 5, -5, 5])

    # Adding colorbar
    cbar = plt.colorbar(label='Perda em log')
    tick_locations = [0.001, 0.00316, 0.01, 0.0316, 0.1, 0.316, 1]
    tick_labels = ['0.001', '0.00316', '0.01', '0.0316', '0.1', '0.316', '1']
    cbar.set_ticks(tick_locations)
    cbar.set_ticklabels(tick_labels)

    # Ading title and labels
    plt.title(f'etch depth :{etch_depths[i]} nm | lambda = 1550 nm')
    plt.xlabel('x')
    plt.ylabel('y')

    # Save images
    file_path = os.path.join(save_path, f'frame_{i}.png')
    plt.savefig(file_path)
    plt.close()

    # Load GIF as frame
    frame = Image.open(file_path)
    frames.append(frame)

# Create GIF and save it on the PC
gif_path = os.path.join(save_path, 'animation.gif')
frames[0].save(gif_path, save_all=True, append_images=frames[1:], optimize=True,
duration=50, loop=0)

# Showing gif in code
display(IPImage(filename=gif_path))

# showing path where GIF was saved in the pc
print(f"GIF saved in: {gif_path}")
```

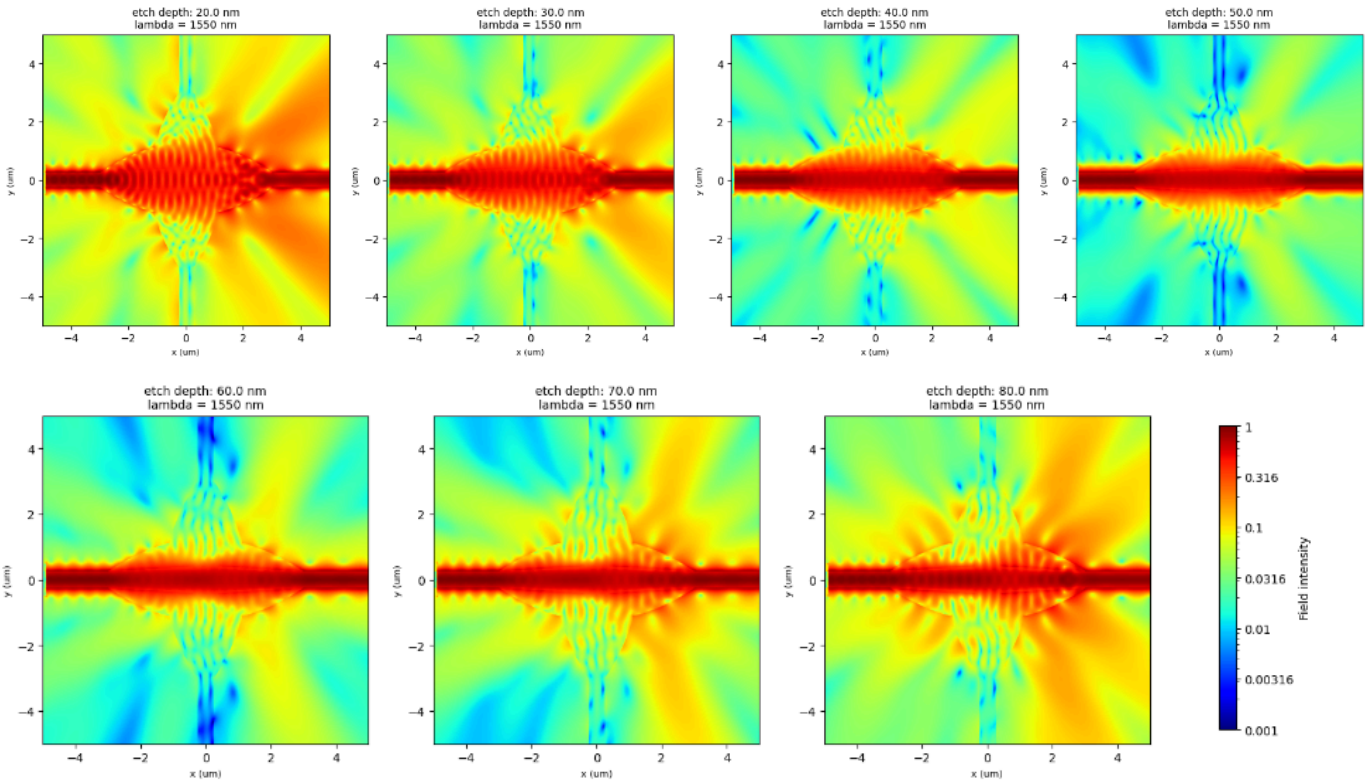



```
fig, axs = plt.subplots(1, 7, figsize=(40, 5) )

for i in range(7):
    im = axs[i].imshow(heatmap_mag[:, :, 9,  (i*int((T_data.shape[1]-1)/6))],
cmap='jet', vmin=0.001, vmax=1,extent=[-5, 5, -5, 5])

    axs[i].set_title(f'etch depth: {etch_depth[i]:.1f} nm\nlambda = 1550 nm',
fontsize=10)
    axs[i].set_xlabel('x (um)', fontsize=8)
    axs[i].set_ylabel('y (um)', fontsize=8)

cbar_ax = fig.add_axes([0.92, 0.15, 0.005, 0.7])  # [x, y, largura, altura]
cbar = fig.colorbar(im, cax=cbar_ax)
cbar.set_label('Field intensity')
plt.show()
```



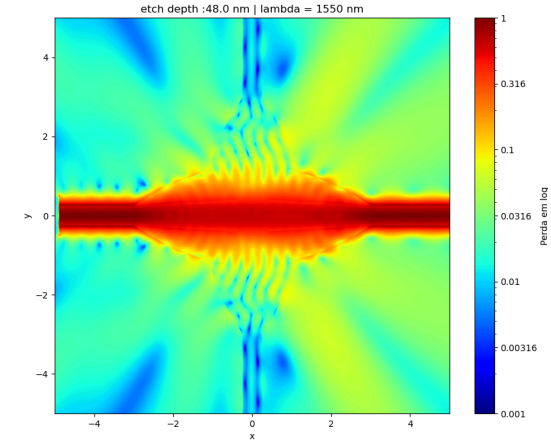


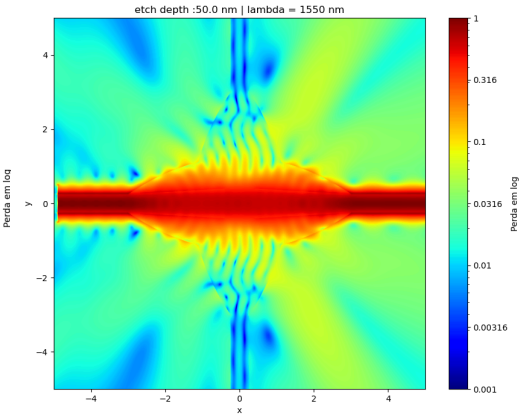

```
fig, axs = plt.subplots(1, 7, figsize=(40, 5))

for i in range(7):
    im = axs[i].imshow(heatmap_mag[:, :, 9,  (i*int((T_data.shape[1]-1)/6))],
cmap='jet', norm=colors.LogNorm(vmin=0.001, vmax=1),
                       extent=[-5, 5, -5, 5])

    axs[i].set_title(f'etch depth: {etch_depth[i]:.1f} nm\nlambda = 1550 nm',
fontsize=10)
    axs[i].set_xlabel('x (um)', fontsize=8)
    axs[i].set_ylabel('y (um)', fontsize=8)

cbar_ax = fig.add_axes([0.92, 0.15, 0.005, 0.7])
cbar = fig.colorbar(im, cax=cbar_ax)
cbar.set_label('Field intensity')
tick_locations = [0.001, 0.00316, 0.01, 0.0316, 0.1, 0.316, 1]
tick_labels = ['0.001', '0.00316', '0.01', '0.0316', '0.1', '0.316', '1']
cbar.set_ticks(tick_locations)
cbar.set_ticklabels(tick_labels)

plt.show()
```

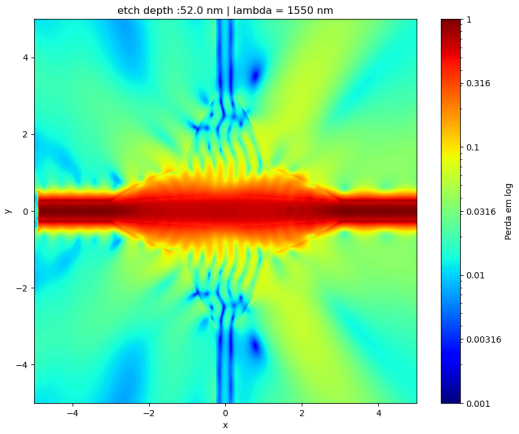


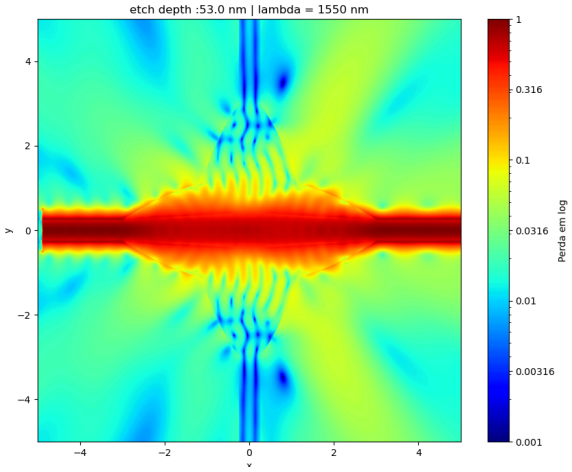
Com os resultados plotados em formato em gif podemos ver a progressão da perda de acordo com a profundidade do etch, e com as imagens estáticas podemos confirmar que a perda mais baixa é sim ao redor de 50nm.

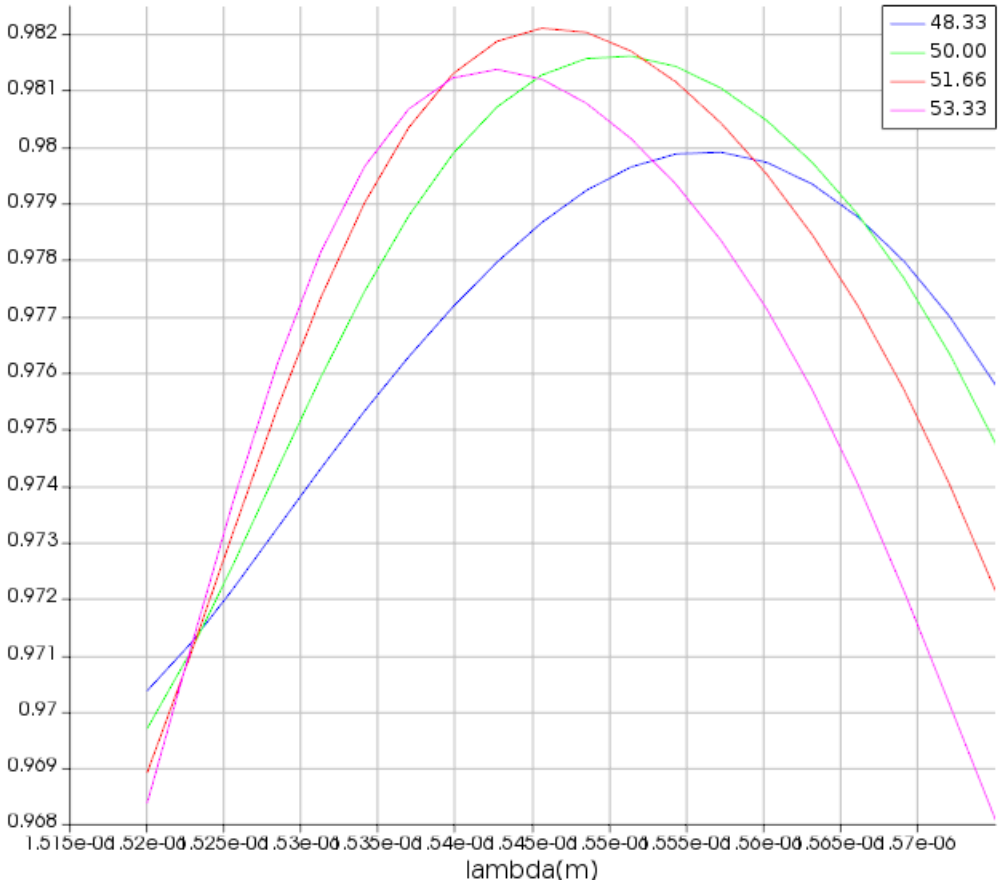
Observando os resultados ao redor de 50 nm mais precisamente podemos ver que 52

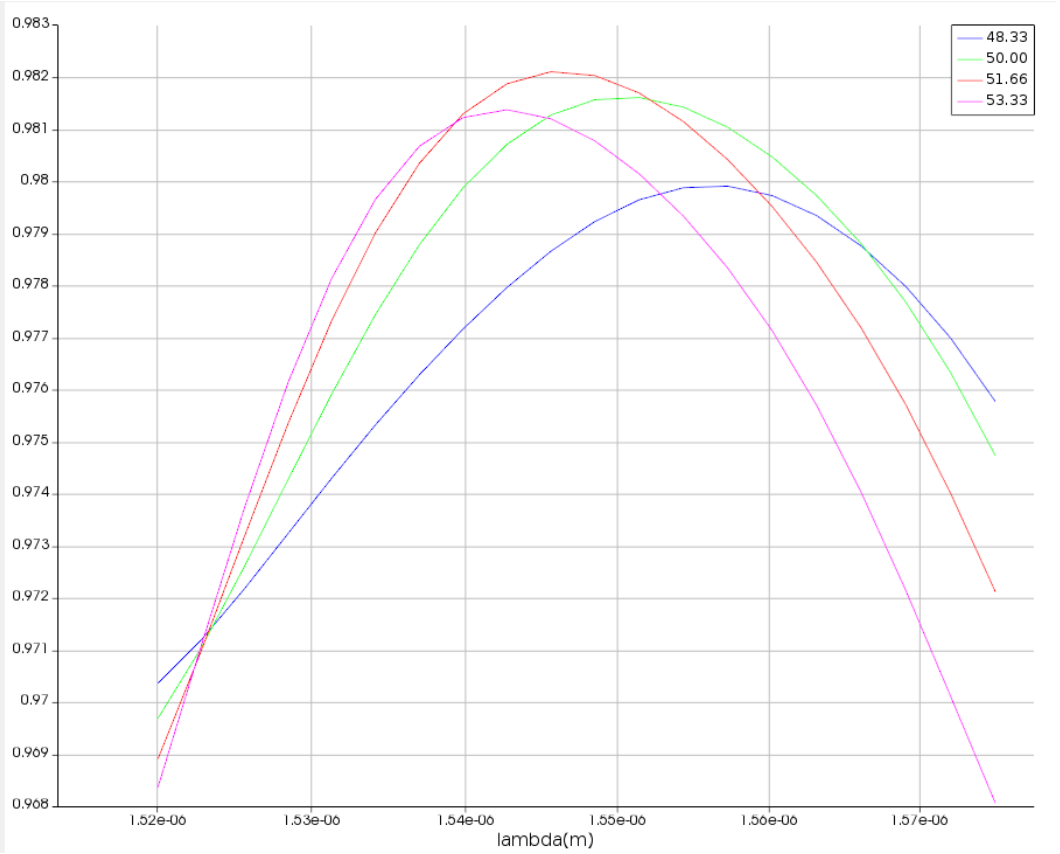












Citações

(1) Bogaerts, Wim & Dumon, Pieter & Thourhout, Dries & Baets, Roel. (2007). Low-loss, low-cross-talk crossings for silicon-on-insulator nanophotonic waveguides. Optics Letters. 32. 2801-2803. 10.1364/OL.32.002801.