Blender のバージョンが 2.8x になって，仕様の変更が大きいため，とりあえず動作確認した項目から修正しています.

1 GUI による操作

1.1 オブジェクトの入力と編集

1.2 レンダーリング

1.2.1 境界線を付ける

Render properties の FreeStyle を指定することでレンダーリングの結果に境界線を追加できる.

2 Blender + Phyton

2.1 Python ファイルを指定した Blender の起動方法

test.py に Python プログラムを記述し，以下のように blender を起動する

```
blender -P test.py
```

2.2 初期設定

2.2.1 必要なモジュールの import

```
import os   # オペレーティングシステムの機能を呼び出す
import bpy  # Python から Blender を操作するためのモジュール
import math # 数学関数を使うためのモジュール
```

2.2.2 初期オブジェクトの削除

```
# reset objects
bpy.ops.object.select_all(action='SELECT')
bpy.ops.object.delete(True)
```

2.2.3 World 設定

```
# world sestting
bpy.context.scene.world.node_tree.nodes["Background"].inputs["Color"].default_value = (1,1,1,1)
```

2.2.4 シーンの設定

```
scene = bpy.context.scene
scene.render.resolution_x = 1920  # 解像度
scene.render.resolution_y = 1080  # 解像度
scene.render.resolution_percentage = 100 # 解像度
```

2.2.5 カメラの追加

```
# camera add
bpy.ops.object.camera_add(location=(20.0,-9.0,12.0))
bpy.data.objects['Camera'].rotation_euler = (math.pi*60/180, 0, math.pi*65/180)
```

カメラの方向は X の値を $\pi/2 = 1.57$ とすると地面に平行な方向を向き，少し小さくすることで下方を見下ろすようになる．Z の値を変更することで z 軸を中心にカメラが回転する．Y の値を 0 以外にするとカメラが傾くことになる．

2.2.6 光源の追加

```
# light add
bpy.ops.object.light_add(location=(0.0,0.0,2.0))
```

高原の種類が 2.7x とで変わったように思われる．(詳細は今後調査)

2.2.7 材料の用意

```
# material add
materials = []
materials.append(bpy.data.materials.new('Color0'))
materials[0].diffuse_color = (0,0,1,1)
materials.append(bpy.data.materials.new('Color1'))
materials[1].diffuse_color = (0,1,0,1)
```

2.7x に対して，2.8x では diffuse_color が RGBA での指定になったよう．材料設定の詳細は後で記述する

2.2.8 物体の追加

• 直方体

```
bpy.ops.mesh.primitive_cube_add(location=(0, 0, -0.5)) # 位置を指定して直方体を作成
bpy.ops.transform.resize(value=(5.0,2.0,0.5))          # 直方体の大きさの変更
bpy.context.object.data.materials.append(materials[0]) # 直方体の材料の設定
```

• 円柱

```
bpy.ops.mesh.primitive_cylinder_add(location=(0,0,0),radius=0.5,depth=1.0)
bpy.context.object.data.materials.append(materials[0])
```

• 正多角柱
正多角柱は円柱の頂点を減らすことで生成できる．以下に正六角柱の例を示す.

```
bpy.ops.mesh.primitive_cylinder_add(location=(0,0,0),radius=0.5,depth=0.3,vertices=6)
bpy.context.object.data.materials.append(materials[0])
```

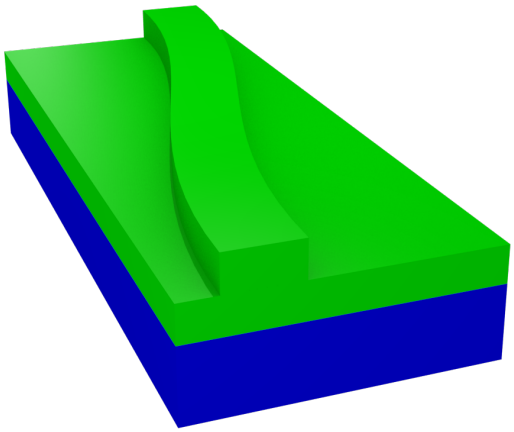
• 多面体
以下に曲がり導波路の例を示す．x 方向の導波路長を 10.0 導波路の幅と高さをそれぞれ 1.0，0.5 とする．また，導波路底面の z 方向位置を 0.0 とし，導波路中心が原点を通るようにしている．

```
nx = 20
Lx = 10.0;
verts = []
faces = []
for i in range (0, nx+1):
    x = Lx/nx*i-5
    y = -0.5+math.cos(math.pi*i/nx)
    z = 0.0
    verts.append((x,y,z))
    y += 1.0
    verts.append((x,y,z))
    z += 0.5
    verts.append((x,y,z))
    y -= 1.0
    verts.append((x,y,z))

for i in range (0, nx): faces.append((4*i+1,4*i+5,4*i+4,4*i+0))
for i in range (0, nx): faces.append((4*i+1,4*i+2,4*i+6,4*i+5))
for i in range (0, nx): faces.append((4*i+2,4*i+3,4*i+7,4*i+6))
for i in range (0, nx): faces.append((4*i+0,4*i+4,4*i+7,4*i+3))
faces.append((0,3,2,1))
faces.append((4*nx,4*nx+1,4*nx+2,4*nx+3))

mesh = bpy.data.meshes.new("Guide")
object = bpy.data.objects.new("Guide", mesh)

object.location = (0,0,0)
bpy.context.scene.collection.objects.link(object)

mesh.from_pydata(verts,[],faces)
```

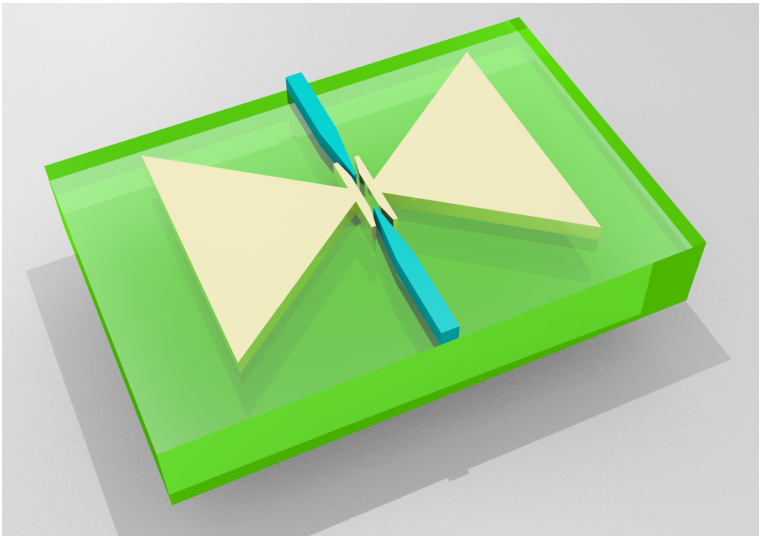
```
mesh.update(calc_edges=True)
mesh.materials.append(materials[1])
```

まず頂点の座標を設定し，頂点から面 (ここでは長方形) を生成する．その後，メッシュオブジェクトを生成し位置を指定しシーンに追加する．このメッシュと先に作成してある頂点，面のデータを関係付け，エッジの情報を更新する．最後に材料の情報を与える．

この実行結果は以下のようになる．ただし，この結果では基板とフィルムを追加している．



同様にして任意の構造を作成できる．以下にボウタイアンテナの作成画像を示す．



2.2.9 オブジェクトのブール演算

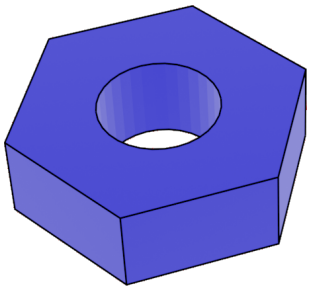
2 つのオブジェクトのブール演算を取ることにより，より複雑な構造を表現できる．

- 物体の差

以下に六角柱から円柱を抜き取ることで三角格子フォトニック結晶の単位セルを生成する例を示す．

```
# ----------------------------------------------------
# ブール演算
def Boolean_DIFF(obj_a, obj_b):
    boolean = obj_a.modifiers.new('My_Bool','BOOLEAN')
    boolean.operation = 'DIFFERENCE'
    boolean.object = obj_b
    bpy.context.view_layer.objects.active = obj_a
    bpy.ops.object.modifier_apply(apply_as='DATA', modifier="My_Bool")
    bpy.data.objects.remove(obj_b)
# ----------------------------------------------------

# 正六角柱の生成
bpy.ops.mesh.primitive_cylinder_add(location=(0,0,0),radius=0.5,depth=0.3,vertices=6)
bpy.context.object.data.materials.append(materials[0])
film_obj = bpy.context.object
# 円柱の生成
bpy.ops.mesh.primitive_cylinder_add(location=(0,0,0),radius=0.2,depth=0.4)
cyl_obj = bpy.context.object
# ブール演算
Boolean_DIFF(film_obj, cyl_obj)
```

モディファイアを適用する前に対象オブジェクトをアクティブにする必要がある．def により，2 つのオブジェクトの差をとる関数を予め作っている．下図は，レンダリングの際に FreeStyle を指定して境界線を付けている．

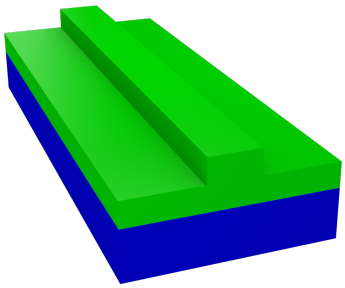


- 物体の結合

  以下に直方体を結合してリブ導波路を作成する例を示す．

```
# 結合のための関数定義
def Boolean_UNION(obj_a, obj_b):
    boolean = obj_a.modifiers.new('My_Bool','BOOLEAN')
    boolean.operation = 'UNION'
    boolean.object = obj_b
    bpy.context.view_layer.objects.active = obj_a
    bpy.ops.object.modifier_apply(apply_as='DATA', modifier="My_Bool")
    bpy.data.objects.remove(obj_b)
# 基板
bpy.ops.mesh.primitive_cube_add(location=(0, 0, -0.5))
bpy.ops.transform.resize(value=(5.0,2.0,0.5))
```

```
bpy.context.object.data.materials.append(materials[0])
# リブ
bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0.25))
bpy.ops.transform.resize(value=(5.0,2.0,0.25))
bpy.context.object.data.materials.append(materials[1])
rib_obj = bpy.context.object
# フィルム
bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0.75))
bpy.ops.transform.resize(value=(5.0,0.5,0.25))
bpy.context.object.data.materials.append(materials[1])
film_obj = bpy.context.object
# リブとフィルムの結合
Boolean_UNION(rib_obj, film_obj)
```

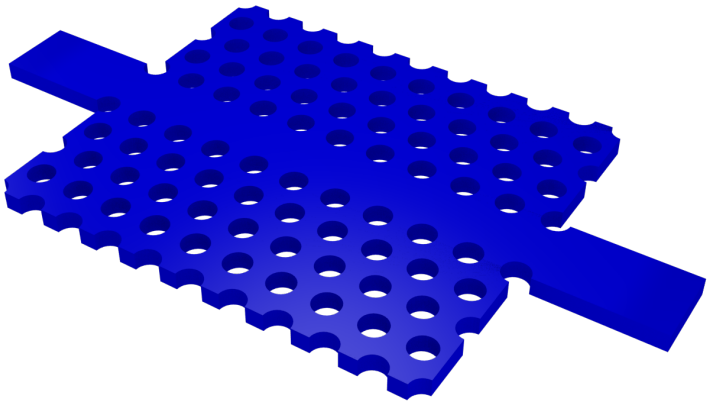


以下にフォトニック結晶導波路の作成例を示す.

```
aa = 1.0
rr = 0.3*aa
ww = aa*math.sqrt(3.0)
WW = 10*aa*math.sqrt(3.0)/2
tt = 0.4*aa
# 長方形スラブ
bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))
bpy.ops.transform.resize(value=(5.0*aa,WW/2,tt/2))
bpy.context.object.data.materials.append(materials[0])
slab_obj = bpy.context.object
# 直線導波路
bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))
bpy.ops.transform.resize(value=(8.0*aa,ww/2,tt/2-0.001))
bpy.context.object.data.materials.append(materials[0])
obj = bpy.context.object
# ブール演算
Boolean_UNION(slab_obj, obj)
for j in range (-5, 6):
    y = j*aa*math.sqrt(3.0)/2
```

```
    if j != 0:
        if j%2 == 0: nm = 1
        else:        nm = 0
        for i in range (-5, 6-nm):
            if j%2 == 0: x = (i+0.5)*aa
            else:        x = i*aa
            bpy.ops.mesh.primitive_cylinder_add(location=(x,y,0),radius=rr,depth=2*tt,vertices=35)
            obj = bpy.context.object
            # ブール演算
            Boolean_DIFF(slab_obj, obj)
```

ここで，直線導波路の高さを (tt/2-0.001) としたり，円筒の分割数を (vertices=35) としているのは，2.8x になって，2 つのオブジェクトの辺や面が完全に一致しているとうまく Bool 演算できないようで，わざとずらしているためである．この実行結果は以下のようになる．



同様にして作成した PCF の画像を以下に示す．材質にガラスを設定している．