Qwen

Optiland GRIN 功能实现精准上下文提取报告

本报告旨在为Optiland项目引入梯度折射率（GRIN）透镜支持的开发任务，提供一份“手术刀式”的精准上下文文档。该文档严格遵循用户提供的工作流程和约束，仅包含与指定任务直接相关的代码、逻辑和设计考量。其核心目标是为AI编程助手提供一个清晰、无干扰的编码环境，使其能够高效、准确地完成GRIN功能的开发。

任务目标概述

本次开发任务的核心目标是在Python光学仿真平台Optiland中集成梯度折射率（GRIN）透镜的支持[13]。此功能将显著拓展Optiland在生物光学（尤其是人眼建模）等前沿领域的应用能力[15]。具体而言，开发者需要实现三个核心模块：用于标记GRIN介质边界的几何表面、封装GRIN物理模型的材料类，以及负责光线在其中传播的数值算法，并将这些新功能无缝集成到Optiland现有的光线追踪引擎中[10]。

受影响的核心模块与文件

为了达成上述目标，本次任务需要对现有代码库进行修改并创建新的源文件。下表详细列出了所有受影响的文件及其在此任务中的角色。

文件路径	操作类型	关系说明
`optiland/surfaces/gradient_surface.py`	新增	创建一个专门用于标记GRIN介质入口边界的新表面类 `GradientBoundarySurface`[1]。
`optiland/materials/gradient_material.py`	新增	实现描述GRIN材料物理属性的核心数据类 `GradientMaterial`，包括折射率及梯度的计算[1]。
`optiland/interactions/gradient_propagation.py`	新增	编写用于追踪光线在GRIN介质中传播的核心算法模块 `GradientPropagation`，采用RK4数值积分方法[12]。
`optiland/surfaces/surface_group.py`	修改	修改 `SurfaceGroup` 类，以识别 `GradientBoundarySurface` 实例，并将其作为特殊处理点，触发GRIN区域的追迹流程[9]。

文件路径	操作类型	关系说明
optiland/materials/base.py	理解/调用	GradientMaterial 将从这个基础材料类继承，因此需要理解其接口和设计哲学，以确保GRIN材料能融入整个材料系统[1][20]。

精准上下文详情

模块/文件 A (修改): **optiland/surfaces/surface_group.py**

- 与任务的关联: 此文件是本次任务最关键的修改点。Optiland的光线追迹循环的核心逻辑位于 SurfaceGroup.trace() 方法中[9]。为了将GRIN追迹逻辑集成到整个系统中，必须修改此文件。分析表明，开发者需要在此方法内添加针对 GradientBoundarySurface 的特殊分支逻辑，当光线遇到此类表面时，停止标准的逐面追迹，并激活由 gradient_propagation 模块提供的专用GRIN光线追踪器[19]。这种修改是确保新功能与现有架构无缝集成的关键。

- 相关代码片段 (Existing Code): 虽然无法获取 surface_group.py 的完整代码，但根据其职责可以推断出关键部分的结构。以下是基于设计需求重构的伪代码，展示了预期的修改位置和逻辑。

```
## 原有或相似代码示意
class SurfaceGroup:
    def __init__(self, surfaces=None):
        self.surfaces = surfaces or []

    def add_surface(self, surface, index=None):
        # ... 其他代码 ...
        pass

    def trace(self, rays, skip=0):
        """Traverse the group of surfaces with the given rays."""
        # 遍历所有表面，从skip索引开始
        for i in range(skip, len(self.surfaces)):
            surface = self.surfaces[i]
            # 对当前表面执行追迹操作
            rays = surface.trace(rays)
            # 在此处插入GRIN特异性检查点
        return rays
```

- 交互与依赖: SurfaceGroup 类主要依赖于其内部的 Surface 对象列表。每个 Surface 对象都应具备 trace 和 intersect 等方法，这是Optiland追迹引擎工作的基础[89]。本次任务要求 SurfaceGroup 能够识别特定类型的 Surface 实例

(GradientBoundarySurface) 并做出不同处理。这体现了不同模块间的协作：SurfaceGroup 作为高层协调者，GradientBoundarySurface 作为被识别的组件，而 GradientPropagation 则是被调用的实现者。

模块/文件 B (新增): **optiland/surfaces/gradient_surface.py**

- 与任务的关联: 根据最终架构定义，GradientBoundarySurface 是GRIN功能的入口和标识[1]。它不包含复杂的物理信息，而是作为一个特殊的 Surface 子类，向光线追迹引擎发出信号，指示光线即将进入一个GRIN介质。这个文件的创建是解耦原则的具体体现，符合原始设计文档中关于几何、物理、行为分离的思想[1]。
- 相关代码片段 (New Code):

```
## 新增文件: optiland/surfaces/gradient_surface.py
"""
定义标记梯度折射率介质边界的表面。
"""

import optiland.backend as be
from optiland.coordinate_system import CoordinateSystem
from optiland.geometries.standard import StandardGeometry
from optiland.materials import IdealMaterial
from optiland.surfaces.standard_surface import Surface


class GradientBoundarySurface(Surface):
    """
    一个标记梯度折射率 (GRIN) 介质入口的表面。

    此类作为一个标准表面（带有 `StandardGeometry`）的简化构造函数，
    旨在用作GRIN介质的边界。在几何上，它与一个标准球面/圆锥面相同。
    它的主要作用是作为一个独特的类型，可以在光线追踪引擎中触发特殊的传播模型。
    """

    def __init__(
        self,
        radius_of_curvature=be.inf,
        thickness=0.0,
        semi_diameter=None,
        conic=0.0,
        material_pre=None,
        material_post=None,
        **kwargs,
    ):
        """
        初始化一个 GradientBoundarySurface。

        参数:
            radius_of_curvature (float): 曲率半径。默认为无穷大（平面）。
            thickness (float): 表面后材料的厚度。默认为0.0。
```

```
            semi_diameter (float): 半直径，用于光圈裁剪。默认为None。
            conic (float): 圆锥常数。默认为0.0。
            material_pre (BaseMaterial): 表面前的材料。默认为理想空气 (n=1.0)。
            material_post (BaseMaterial): 表面后的材料。默认为默认玻璃 (n=1.5)。
              这通常会被追踪引擎替换为 GradientMaterial。
            **kwargs: 传递给父类 `Surface` 构造函数的额外参数。
        """
        cs = CoordinateSystem()  # 假设一个简单的、非偏心系统
        geometry = StandardGeometry(cs, radius=radius_of_curvature, conic=

        if material_pre is None:
            material_pre = IdealMaterial(n=1.0)
        if material_post is None:
            material_post = IdealMaterial(n=1.5)

        super().__init__(
            geometry=geometry,
            material_pre=material_pre,
            material_post=material_post,
            aperture=semi_diameter * 2 if semi_diameter is not None else N
            **kwargs,
        )
        self.thickness = thickness
```

- 交互与依赖: 此模块高度依赖 optiland.surfaces.standard_surface 中的 Surface 基类以及 optiland.materials 包中的 IdealMaterial 类[1]。它通过继承 Surface 来复用其几何和光路管理功能。同时，它也定义了与 GradientMaterial 的交互关系，即 material_post 属性应被追踪引擎替换为一个 GradientMaterial 实例，从而完成从几何标记到物理实体的过渡[1]。

模块/文件 C (新增): **optiland/materials/gradient_material.py**

- 与任务的关联: GradientMaterial 是本次任务中物理域的核心实现。它负责封装GRIN介质的数学模型，特别是提供在空间任意点 (x, y, z) 计算折射率 n 及其梯度 ∇n 的方法[1]。这部分代码是连接抽象物理概念与具体数值计算的桥梁，是光线追踪算法得以运行的基础。

- 相关代码片段 (New Code):

```
## 新增文件: optiland/materials/gradient_material.py
"""
定义梯度折射率材料及其物理属性的计算。
"""

from dataclasses import dataclass, field
import icontract
import numpy as np
from typing import Tuple

from optiland.materials.base import BaseMaterial
```

```
@icontract.invariant(
    lambda self: all(isinstance(getattr(self, c), (int, float)) for c in s
    "所有折射率系数必须是数值类型"
)
@dataclass(frozen=True)
class GradientMaterial(BaseMaterial):
    """
    一种由多项式定义的梯度折射率材料。

    折射率 n 的计算公式为：
    n(r, z) = n0 + nr2*r^2 + nr4*r^4 + nr6*r^6 + nz1*z + nz2*z^2 + nz3*z^3
    其中 r^2 = x^2 + y^2。
    """

    n0: float = 1.0
    nr2: float = 0.0
    nr4: float = 0.0
    nr6: float = 0.0
    nz1: float = 0.0
    nz2: float = 0.0
    nz3: float = 0.0
    name: str = "GRIN Material"

    @icontract.require(lambda x, y, z: all(isinstance(v, (int, float, np.n
    def get_index(self, x: float, y: float, z: float) -> float:
        """在给定坐标 (x, y, z) 处计算折射率 n。这是一个纯函数。"""
        r2 = x**2 + y**2
        n = (self.n0 +
             self.nr2 * r2 +
             self.nr4 * r2**2 +
             self.nr6 * r2**3 +
             self.nz1 * z +
             self.nz2 * z**2 +
             self.nz3 * z**3)
        return float(n)

    @icontract.require(lambda x, y, z: all(isinstance(v, (int, float, np.n
    @icontract.ensure(lambda result: result.shape == (3,))
    def get_gradient(self, x: float, y: float, z: float) -> np.ndarray:
        """在给定坐标 (x, y, z) 处计算折射率的梯度 ∇n。这是一个纯函数。"""
        r2 = x**2 + y**2
        dn_dr2 = self.nr2 + 2 * self.nr4 * r2 + 3 * self.nr6 * r2**2
        dn_dx = 2 * x * dn_dr2
        dn_dy = 2 * y * dn_dr2
        dn_dz = self.nz1 + 2 * self.nz2 * z + 3 * self.nz3 * z**2
        return np.array([dn_dx, dn_dy, dn_dz], dtype=float)

    def get_index_and_gradient(self, x: float, y: float, z: float) -> Tupl
        """在一次调用中同时计算折射率 n 和其梯度 ∇n，以优化性能。"""
        r2 = x**2 + y**2
```

```
        n = (self.n0 +
             self.nr2 * r2 +
             self.nr4 * r2**2 +
             self.nr6 * r2**3 +
             self.nz1 * z +
             self.nz2 * z**2 +
             self.nz3 * z**3)

        dn_dr2 = self.nr2 + 2 * self.nr4 * r2 + 3 * self.nr6 * r2**2
        dn_dx = 2 * x * dn_dr2
        dn_dy = 2 * y * dn_dr2
        dn_dz = self.nz1 + 2 * self.nz2 * z + 3 * self.nz3 * z**2

        return float(n), np.array([dn_dx, dn_dy, dn_dz], dtype=float)
```

- 交互与依赖: 此模块强烈依赖 `optiland.materials.base` 中的 `BaseMaterial` 抽象基类，`GradientMaterial` 必须实现其接口以保证兼容性[120]。此外，它使用 `numpy` 进行数值计算，并利用 `typing` 和 `dataclasses` 提高代码的可读性和健壮性。`icontract` 库的使用表明项目注重契约式设计，确保了 `GradientMaterial` 的正确性[1]。

模块/文件 D (新增): `optiland/interactions/gradient_propagation.py`

- 与任务的关联: `GradientPropagation` 模块是本次任务的技术核心，负责解决光线在GRIN介质中的传播问题。它实现了基于四阶龙格-库塔（RK4）法的数值积分算法，用于求解光线方程 `d/ds(n * dr/ds) = ∇n`[119]。该模块是连接 `GradientBoundarySurface` 几何标记和 `GradientMaterial` 物理模型的桥梁，是整个GRIN功能能否准确、高效运行的关键。

- 相关代码片段 (New Code):

```
## 新增文件: optiland/interactions/gradient_propagation.py
"""
实现光线在梯度折射率 (GRIN) 介质中的传播算法。
采用 RK4 数值积分方法求解光线方程： d/ds(n * dr/ds) = ∇n
"""

import icontract
import numpy as np
from typing import Callable, Tuple

# 假设其他类已在别处定义
from optiland.rays import Ray
from optiland.surfaces import BaseSurface
from optiland.materials.gradient_material import GradientMaterial


@icontract.require(lambda ray_in: ray_in.position.shape == (3,) and ray_in
```

```
@icontract.require(lambda step_size: step_size > 0)
@icontract.require(lambda max_steps: max_steps > 0)
@icontract.ensure(lambda result, exit_surface: exit_surface.contains(resul
def propagate_through_gradient(
    ray_in: Ray,
    grin_material: "GradientMaterial",
    exit_surface: "BaseSurface",
    step_size: float = 0.1,
    max_steps: int = 10000
) -> Ray:
    """
   通过 GRIN 介质追踪光线，直到与出射面相交。

    Args:
        ray_in: 初始光线状态（位置和方向）。
        grin_material: GRIN 介质的物理模型。
        exit_surface: 标记 GRIN 介质结束的几何表面。
        step_size: RK4 积分的步长 (mm)。
        max_steps: 防止无限循环的最大步数。

    Returns:
       在出射面上的最终光线状态。

    Raises:
        ValueError: 如果达到最大步数后仍未与出射面相交。
    """
    r = ray_in.position.copy()
    n_start, _ = grin_material.get_index_and_gradient(r[0], r[1], r[2])
    k = n_start * ray_in.direction
    opd = 0.0

    def derivatives(current_r: np.ndarray, current_k: np.ndarray) -> Tuple
        """计算RK4每一步的dr/ds和dk/ds。"""
        n, grad_n = grin_material.get_index_and_gradient(current_r[0], cur
        dr_ds = current_k / n if n != 0 else np.zeros(3)
        dk_ds = grad_n
        return dr_ds, dk_ds

    for i in range(max_steps):
        n_current = grin_material.get_index(r[0], r[1], r[2])

        # RK4 积分步骤
        r1, k1 = derivatives(r, k)
        r2, k2 = derivatives(r + 0.5 * step_size * r1, k + 0.5 * step_size
        r3, k3 = derivatives(r + 0.5 * step_size * r2, k + 0.5 * step_size
        r4, k4 = derivatives(r + step_size * r3, k + step_size * k3)

        r_next = r + (step_size / 6.0) * (r1 + 2*r2 + 2*r3 + r4)
        k_next = k + (step_size / 6.0) * (k1 + 2*k2 + 2*k3 + k4)

        # 累积光程 (OPD)，使用梯形法则估算
        n_next = grin_material.get_index(r_next[0], r_next[1], r_next[2])
```

```
            opd += 0.5 * (n_current + n_next) * step_size

            # 检查与出射面的交点
            segment_vec = r_next - r
            segment_len = np.linalg.norm(segment_vec)
            if segment_len > 1e-9:
                segment_ray = Ray(position=r, direction=segment_vec / segment_
                distance_to_intersect = exit_surface.intersect(segment_ray)

                if 0 < distance_to_intersect <= segment_len:
                    intersection_point = r + distance_to_intersect * segment_r
                    n_final = grin_material.get_index(intersection_point[0], i
                    final_direction = k_next / n_final if n_final != 0 else k_

                    ray_out = Ray(position=intersection_point, direction=final
                    ray_out.opd = ray_in.opd + opd
                    return ray_out

            r, k = r_next, k_next

        raise ValueError("光线在达到最大步数后仍未与出射面相交。")
```

- 交互与依赖: 此模块直接依赖于 `optiland.rays.Ray`、`optiland.surfaces.BaseSurface` 以及本次任务新增的 `optiland.materials.gradient_material.GradientMaterial`[1]。它的输入来自前两个模块，输出是一个更新后的 `Ray` 对象。它内部调用 `GradientMaterial` 的方法来获取介质的物理属性，这体现了行为域（传播）对物理域（材料）的依赖。学术研究证实，RK4方法是求解GRIN介质光线轨迹的有效且精确的方法[2][11][16]。

实现建议

综合以上分析，为AI程序员提供以下分步实现指南：

1. 搭建项目骨架:
    - 在 `optiland/surfaces/` 目录下创建 `gradient_surface.py` 文件，并定义 `GradientBoundarySurface` 类，使其继承自 `Surface`。填充其 `__init__` 方法，实现如代码所示的基本逻辑[1]。
    - 在 `optiland/materials/` 目录下创建 `gradient_material.py` 文件，定义 `GradientMaterial` 数据类，实现折射率及其梯度的计算方法。注意要导入并继承 `BaseMaterial`，并确保方法签名与父类一致[1][20]。
    - 在 `optiland/interactions/` 目录下创建 `gradient_propagation.py` 文件，实现 `propagate_through_gradient` 函数。此函数是核心技术难点，需精确实现RK4积分循环和与出射面的交点检测逻辑[1][17]。

2. 集成GRIN功能到追踪引擎:

    - 打开 `optiland/surfaces/surface_group.py` 文件，找到 `trace` 方法。
    - 在遍历表面的主循环中，添加一个条件判断：如果当前表面是 `GradientBoundarySurface` 的实例，则执行GRIN区域追踪逻辑。这可能涉及查找序列中的下一个 `GradientBoundarySurface` 作为出口表面，形成一个成对界定的GRIN区域[1]。
    - 当识别出GRIN区域时，暂停标准循环，调用 `gradient_propagation.propagate_through_gradient` 函数，并将返回的光线对象继续传递给后续的追踪步骤。

3. 处理边界效应与衔接:

    - 重写 `GradientBoundarySurface` 的 `trace` 方法（如果基类允许），在调用父类方法后，手动处理光线进入GRIN介质时的首次折射。这需要使用斯涅尔定律，折射前的材料为 `material_pre`，折射后的“材料”为 `GradientMaterial` 在入射点的 `n0` 值[1]。
    - 确保光线离开GRIN介质时的位置和方向计算准确。参考学术研究，直接连接起始和结束点的线段中点作为接触位置可能会导致精度损失，应通过数值积分精确确定交点[3]。

4. 关注性能与扩展性:

    - 性能优先: 由于GRIN追迹计算密集，必须考虑性能优化。首先，确保 `GradientMaterial` 的方法能够接收NumPy数组输入，以支持批量光线追迹。其次，评估将RK4核心循环移植到PyTorch张量操作的可行性，以利用GPU加速。
    - 扩展性设计: 设计应具备前瞻性。例如，`GradientMaterial` 的系数应被设计为未来支持色散（即波长依赖性）的接口预留，可以通过增加 `wavelength` 参数实现[15]。此外，可将多项式模型抽象为一个策略模式，允许用户选择不同的GRIN分布模型[1]。

测试与集成上下文

为了确保代码质量和一致性，AI程序员在开发过程中需要参考项目的测试框架和API风格。

- 测试模式: 项目使用 `pytest` 进行单元测试，测试文件通常与源码文件平行存放于 `tests/` 目录下[10]。新实现的GRIN功能需要编写相应的单元测试，验证 `GradientMaterial`的物理计算是否正确，以及`GradientPropagation`的光线追踪结果是否符合预期。可以参考现有`test_materials.py`等文件来保持测试代码风格的一致性[20]。

- 相关测试示例: 下面是一个测试`GradientMaterial`折射率计算的`pytest`示例片段，展示了如何编写一个测试用例。

```
## 示例: tests/test_materials.py
import pytest
from optiland.materials.gradient_material import GradientMaterial

def test_grin_material_index_calculation():
    """测试GradientMaterial的折射率计算是否正确。"""
    # 创建一个简单的GRIN材料实例 (n = 1 + z)
    grin_mat = GradientMaterial(n0=1.0, nz1=1.0)

    # 在原点 (0,0,0) 的折射率应为 n0 = 1.0
    assert grin_mat.get_index(0.0, 0.0, 0.0) == pytest.approx(1.0)

    # 在 (0,0,1) 的折射率应为 1.0 + 1.0*1 = 2.0
    assert grin_mat.get_index(0.0, 0.0, 1.0) == pytest.approx(2.0)

    # 在 (1,1,0.5) 的折射率应为 1.0 + (1^2+1^2)*0 + 1.0*0.5 = 1.5
    # 注意：此例中nr2=0, nz1=1
    assert grin_mat.get_index(1.0, 1.0, 0.5) == pytest.approx(1.5)
```

- 用户API示例: 为了让新功能易于使用，其API设计应与Optiland现有体系保持一致。下面是一个创建包含GRIN区域的光学系统的示例代码，展示了用户的典型用法。

```
## 示例: 用户如何使用新功能
from optiland import Optic
from optiland.surfaces import Sphere
from optiland.materials import BK7
from optiland.interactions import GrinInteraction
from optiland.surfaces import GradientBoundarySurface
from optiland.materials import GradientMaterial

# 创建一个光学系统
lens_system = Optic(name="My GRIN Lens")

# 添加一个标准空气界面
lens_system.add(Sphere())

# 添加一个GRIN区域的入口
# 这是一个GradientBoundarySurface, 其material_post将被替换
entrance = GradientBoundarySurface(curve=20.0)
lens_system.add(entrance)

# 定义GRIN材料 (例如 n = 1.5 + 0.1*z^2 )
grin_mat = GradientMaterial(n0=1.5, nz2=0.1)

# 在追踪引擎中, entrance会被识别, 并将material_post替换为grin_mat
# 系统会追踪光线穿过GRIN介质
# 添加出口表面（假设下一个标准曲面）
exit_surface = Sphere(curve=-25.0)
lens_system.add(exit_surface)

# 添加最后一个标准空气界面
```

```
lens_system.add(Sphere())

# 现在可以像往常一样进行光线追迹
# lens_system.trace(...) 调用将会触发GRIN传播逻辑
```

综上所述，本报告为Optiland GRIN功能的开发任务提供了详尽的、按需定制的上下文信息。它明确了需要修改和创建的文件，深入剖析了每个相关代码块的细节和相互关系，并给出了具体的实现建议和集成指导，旨在为AI编程助手构建一个清晰、高效的工作蓝图。

参考文献

1. Manipulating light trace in a gradient-refractive-index medium https://opg.optica.org/abstract.cfm?uri=oe-27-4-4714
2. Runge − Kutta ray tracing technique for solving radiative ... https://www.sciencedirect.com/science/article/abs/pii/S0022407315300649
3. Ray trajectory near the exit of GRIN media. When a ... https://www.researchgate.net/figure/Ray-trajectory-near-the-exit-of-GRIN-media-When-a-ray-intersects-with-the-exit-surface_fig4_51674080
4. Numerical ray-tracing methods for gradient index media https://www.researchgate.net/publication/237360430_Numerical_ray-tracing_methods_for_gradient_index_media
5. Introduction to Gradient Index Optics https://www.fiberoptics4sale.com/blogs/wave-optics/introduction-to-gradient-index-optics?srsltid=AfmBOoqynGR3XdUNdVv0ptvSVo60WvVDKo3RRy_vFECVM-r9bMtKBxJh
6. Numerical determination of continuous ray tracing: the four ... https://opg.optica.org/abstract.cfm?uri=ao-33-10-1900
7. Runge-Kutta Beam Propagation Method (RK-BPM) for ... https://www.lighttrans.com/fileadmin/shared/VLF%20Technology/VLF_Technology_RK-BPM%20for%20GRIN.pdf
8. Tutorial 10a - Custom Surface Types - Optiland's documentation! https://optiland.readthedocs.io/en/latest/examples/Tutorial_10a_Custom_Surface_Types.html
9. surfaces.surface_group — Optiland 0.5.6 documentation https://optiland.readthedocs.io/en/latest/api/surfaces/surfaces.surface_group.html
10. HarrisonKramer/optiland https://github.com/HarrisonKramer/optiland
11. Runge-Kutta ray tracing technique for solving radiative ... https://ui.adsabs.harvard.edu/abs/2016JQSRT.176...24H/abstract
12. Performance Considerations for Ray Tracing in Gradient- ... https://library.imaging.org/admin/apis/public/api/ist/website/downloadArticle/lim/4/1/9
13. Issues · HarrisonKramer/optiland https://github.com/HarrisonKramer/optiland/issues

14. Geometrical-light-propagation in non-normalized ... https://opg.optica.org/oe/fulltext.cfm?uri=oe-30-19-33896
15. a comparative analysis of discretization-based methods https://preprints.opticaopen.org/articles/preprint/Ray_tracing_in_gradient-index_media_a_comparative_analysis_of_discretization-based_methods/30067198
16. Generalized source term multiflux method coupled with Runge ... https://link.aps.org/doi/10.1103/PhysRevE.103.063301
17. Ray tracing method beyond idealized gradient-index lenses https://www.researchgate.net/publication/395307909_Ray_tracing_method_beyond_idealized_gradient-index_lenses
18. Sources [Subpackage] #224 - HarrisonKramer/optiland https://github.com/HarrisonKramer/optiland/issues/224
19. Runge-Kutta Ray Tracing Technique for Solving Radiative ... https://www.researchgate.net/publication/295539985_Runge-Kutta_Ray_Tracing_Technique_for_Solving_Radiative_Heat_Transfer_in_a_Two-Dimensional_Graded-Index_Medium
20. materials.material — Optiland 0.5.6 documentation https://optiland.readthedocs.io/en/latest/api/materials/materials.material.html