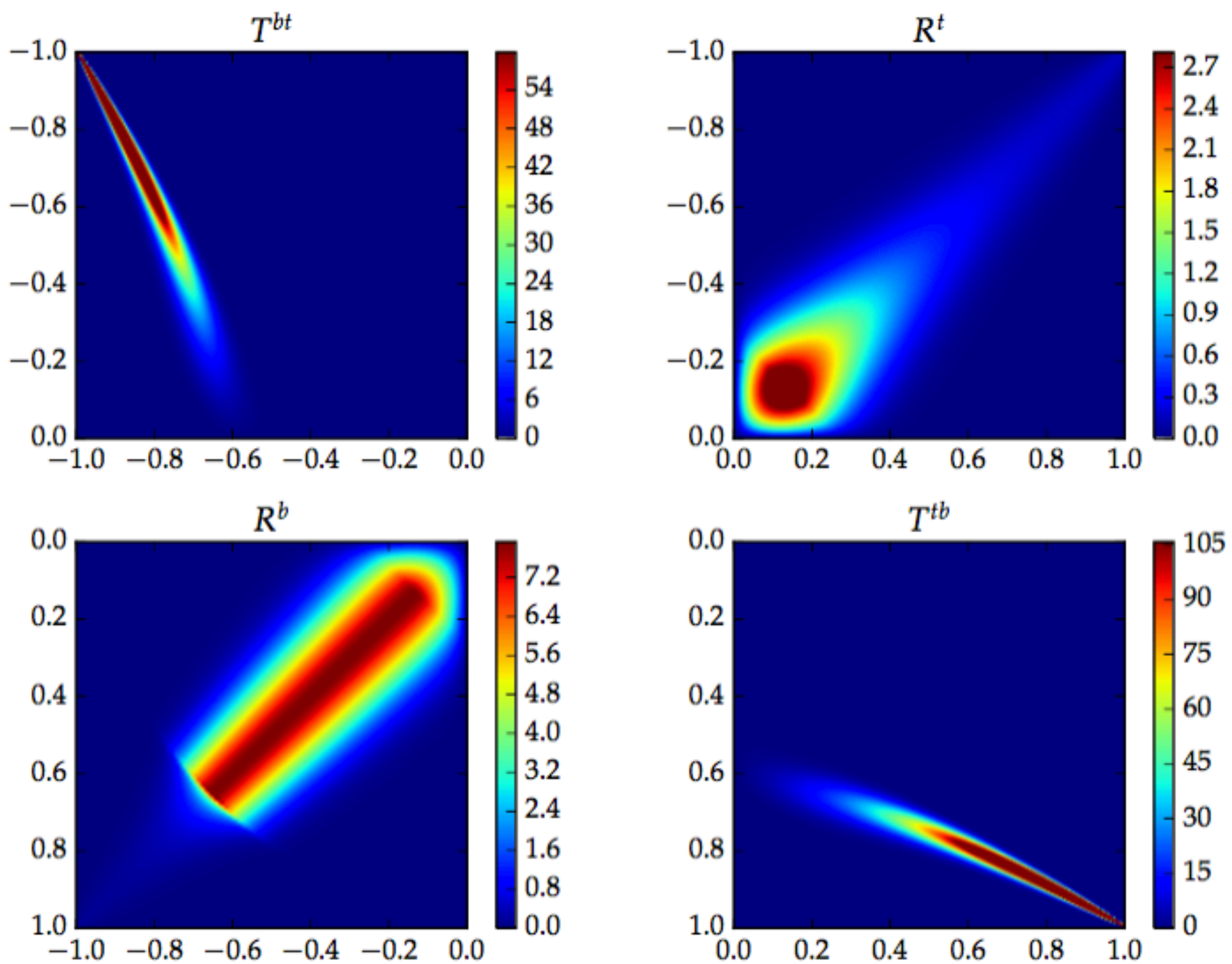
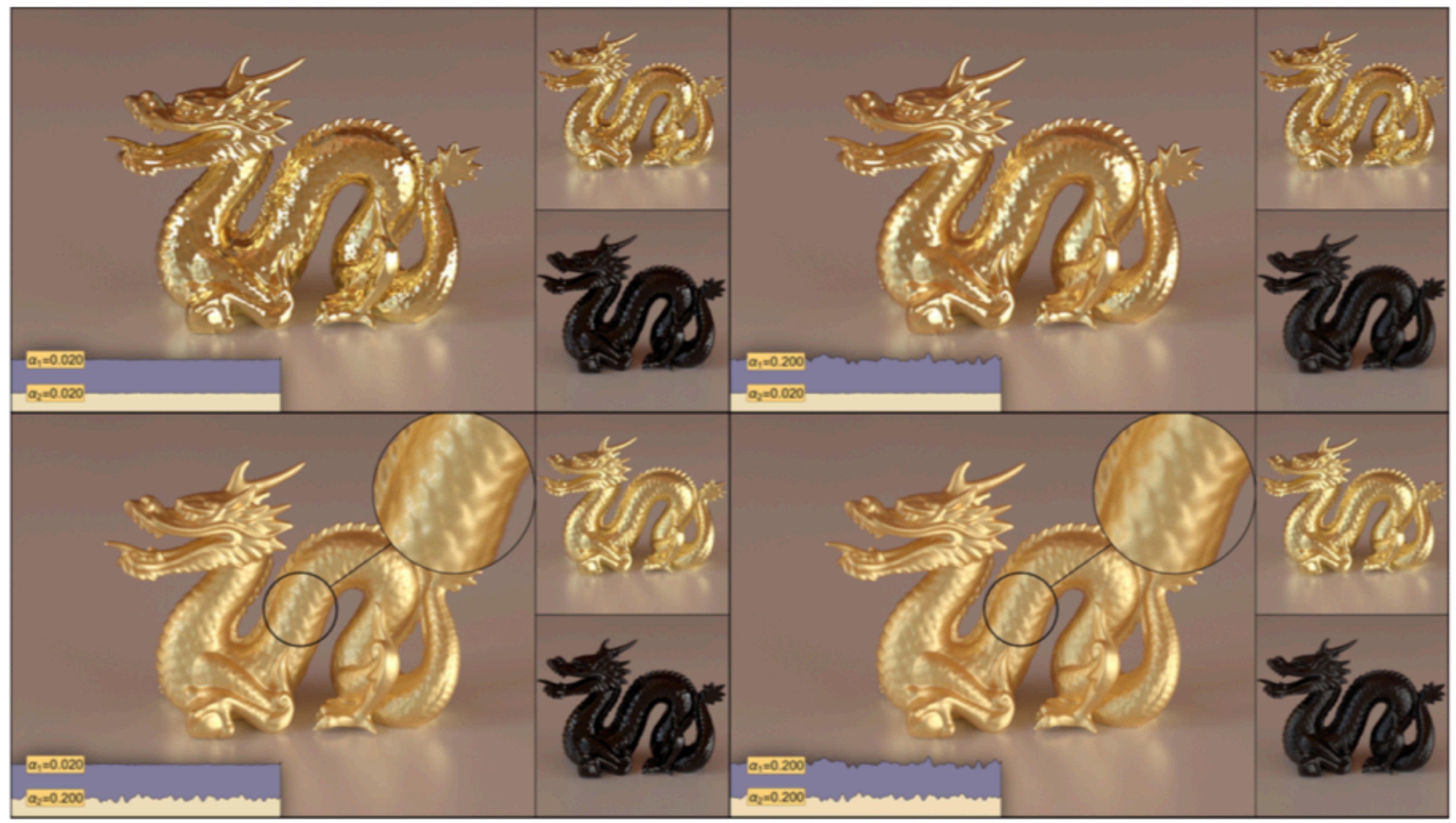
`layerlab`: A computational toolbox for layered materials

Wenzel Jakob

ETH Zürich

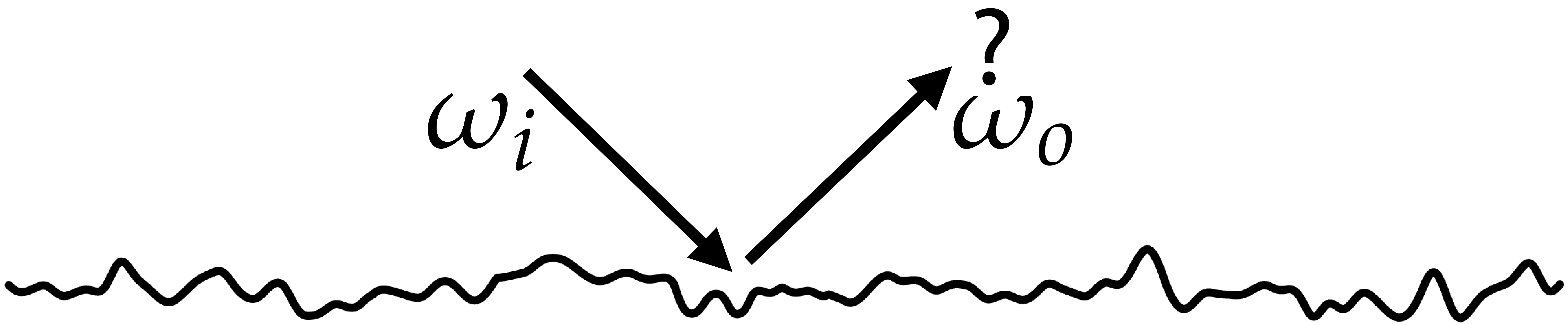




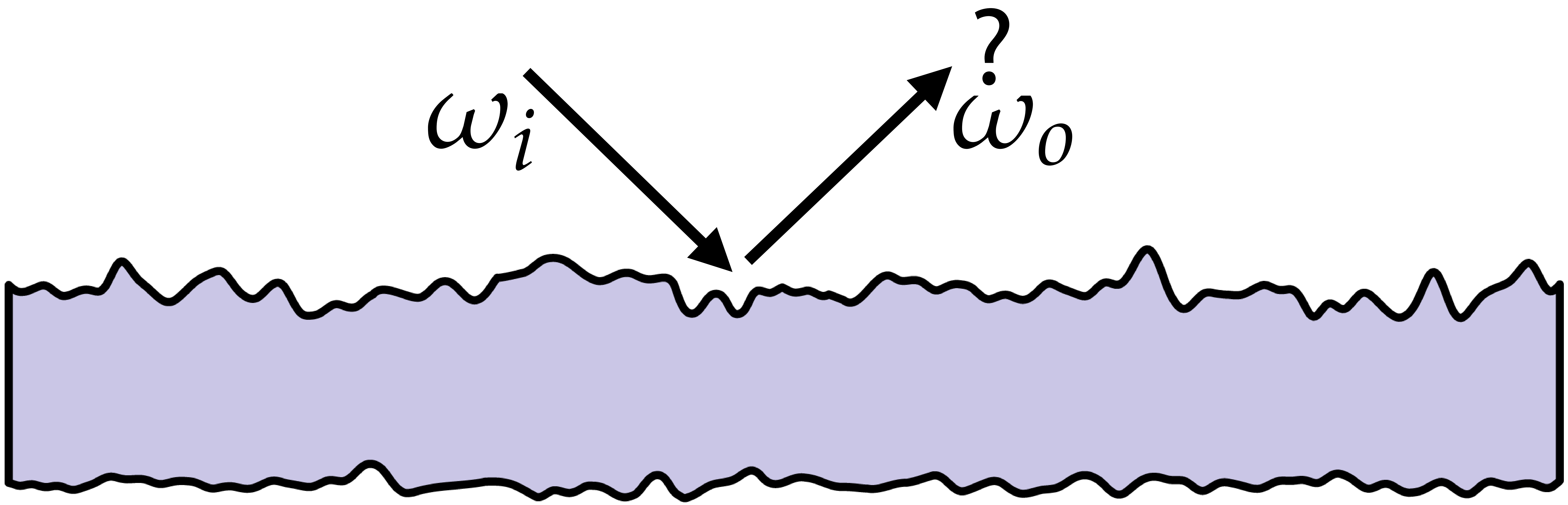
Outline

- What is layerlab?
- Benefits of layered material models
- Layers in flatland
- Combining layers
- Layers in 3D
- Experiments

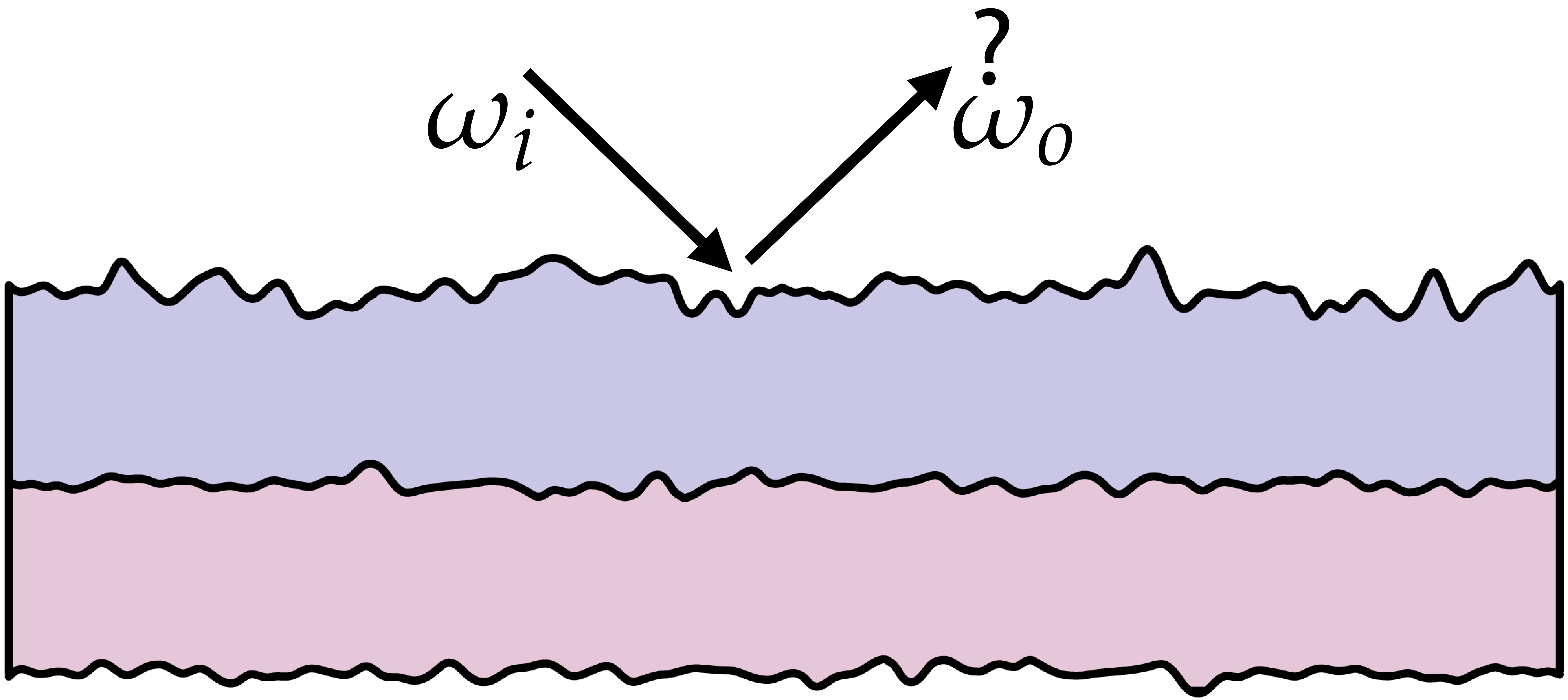
Layered reflectance models



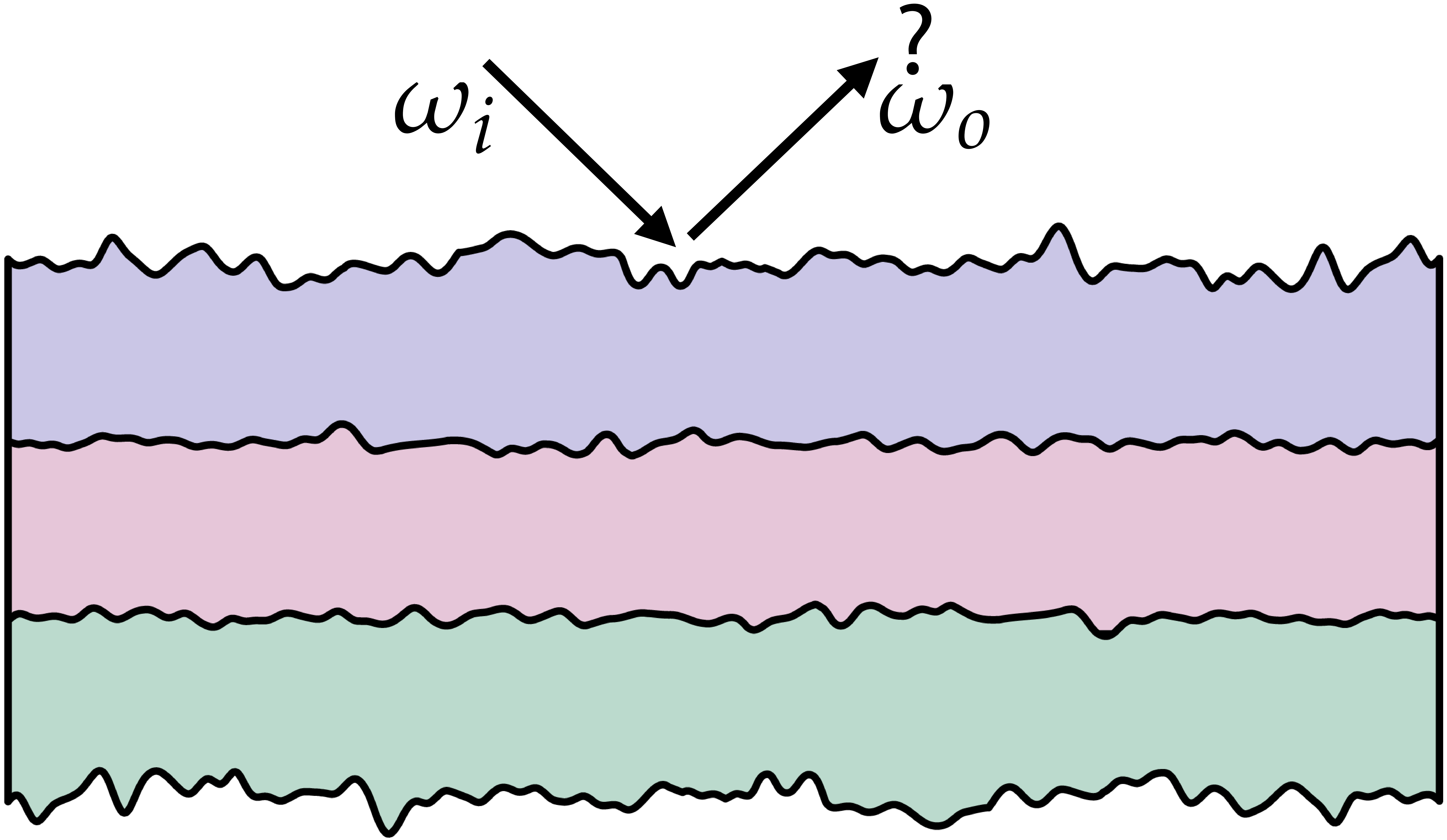
Layered reflectance models



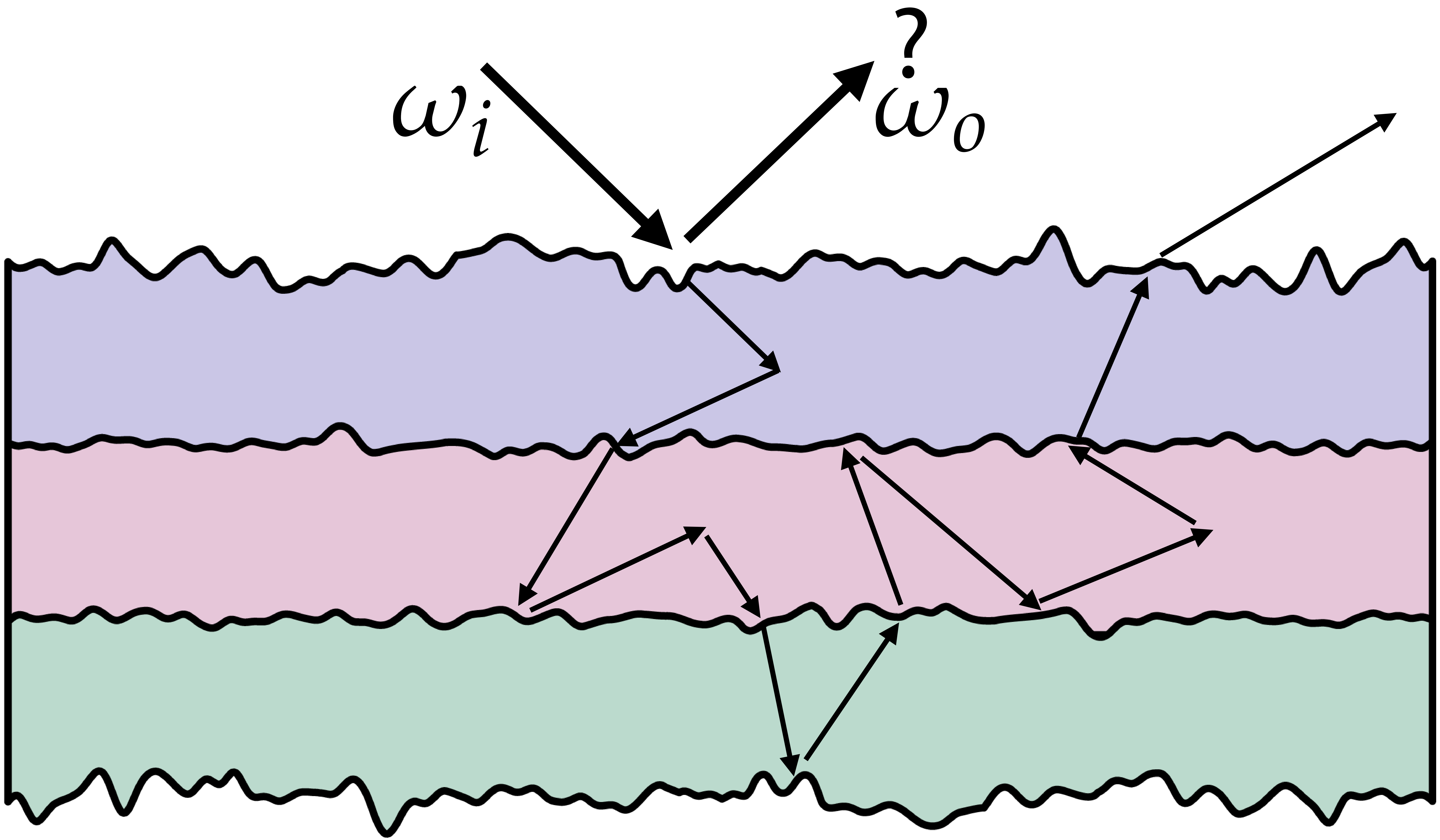
Layered reflectance models



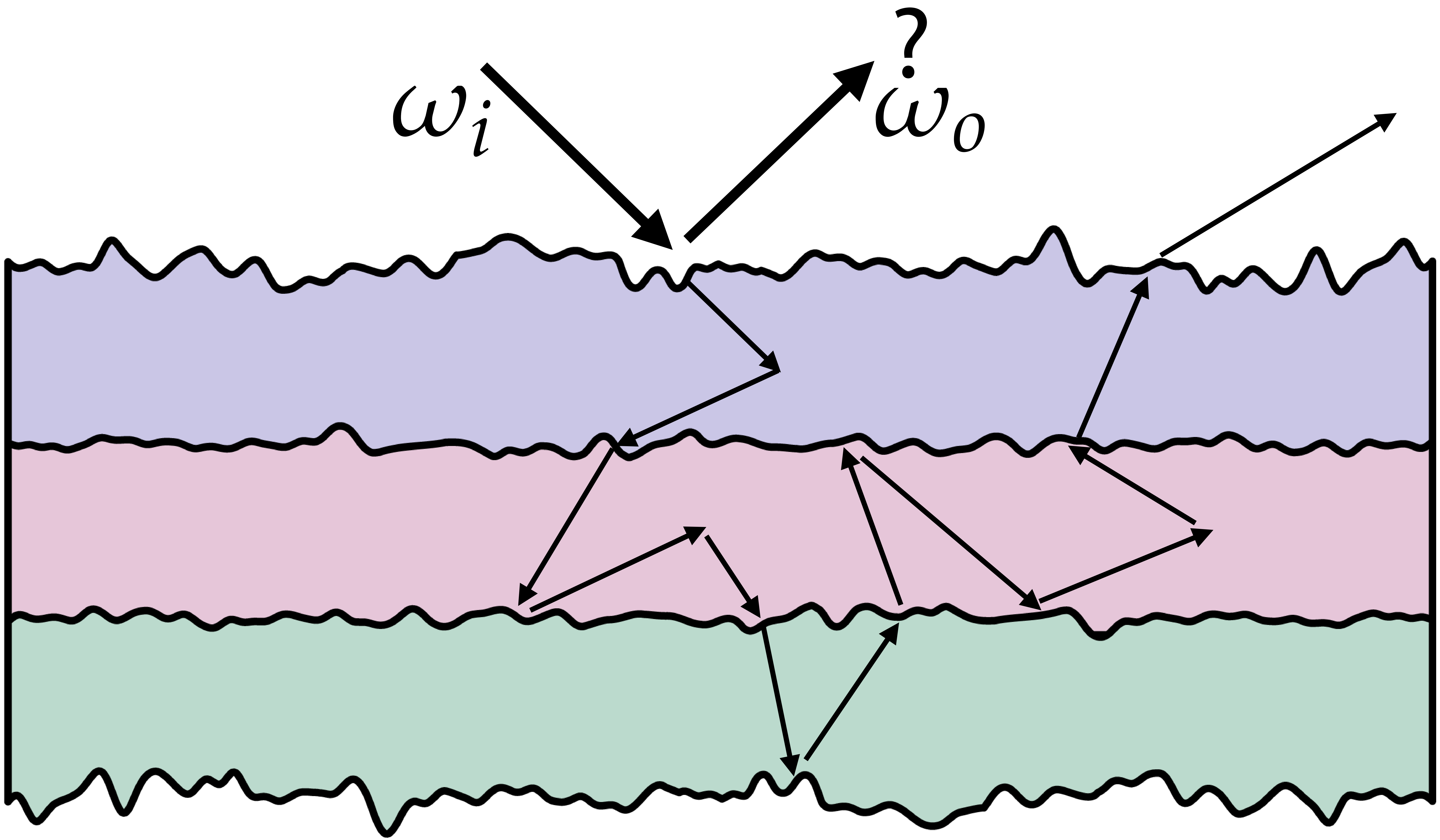
Layered reflectance models



Layered reflectance models



Layered reflectance models

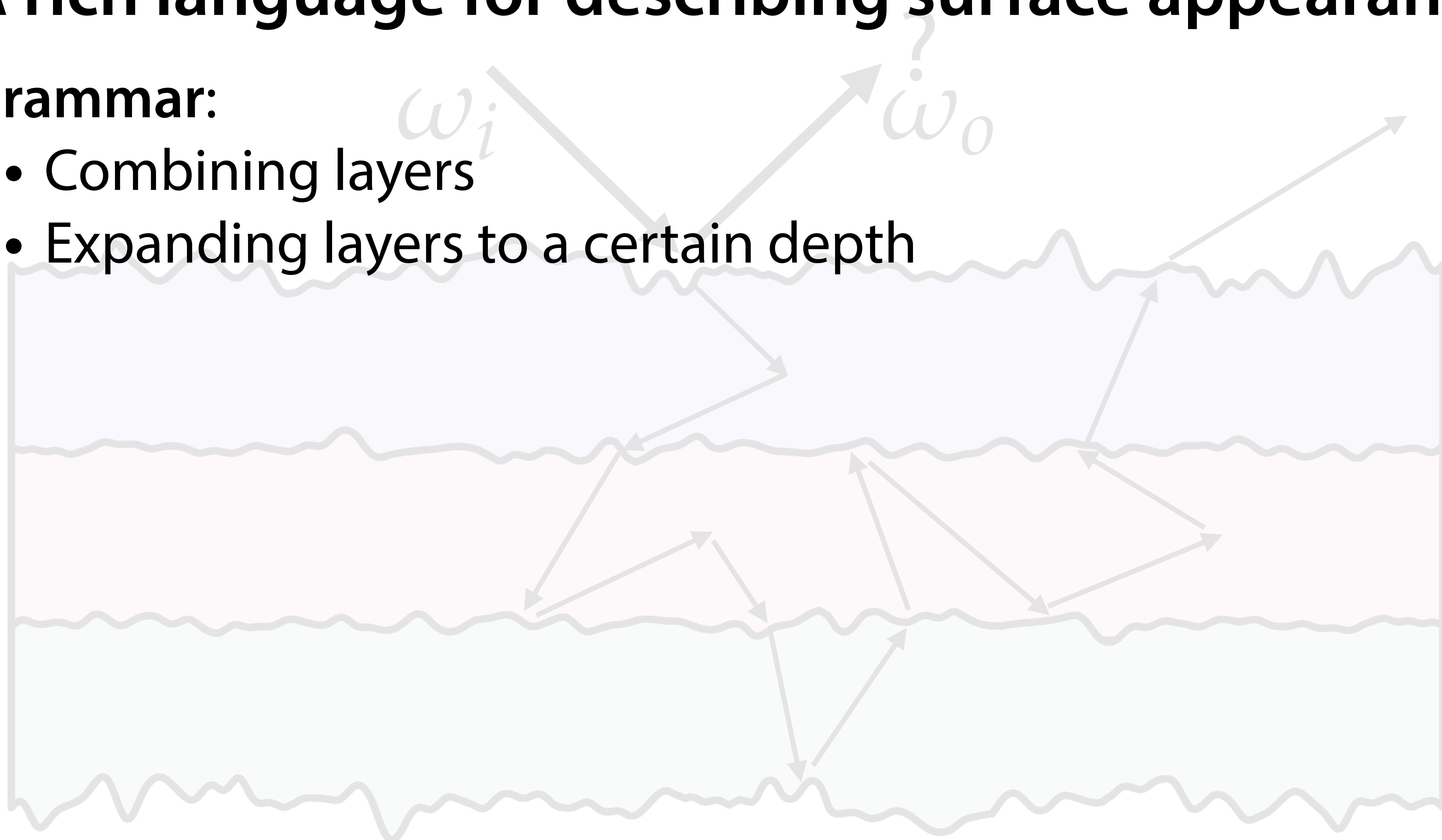


Layered reflectance models

A rich language for describing surface appearance

Grammar:

- Combining layers
- Expanding layers to a certain depth



Layered reflectance models

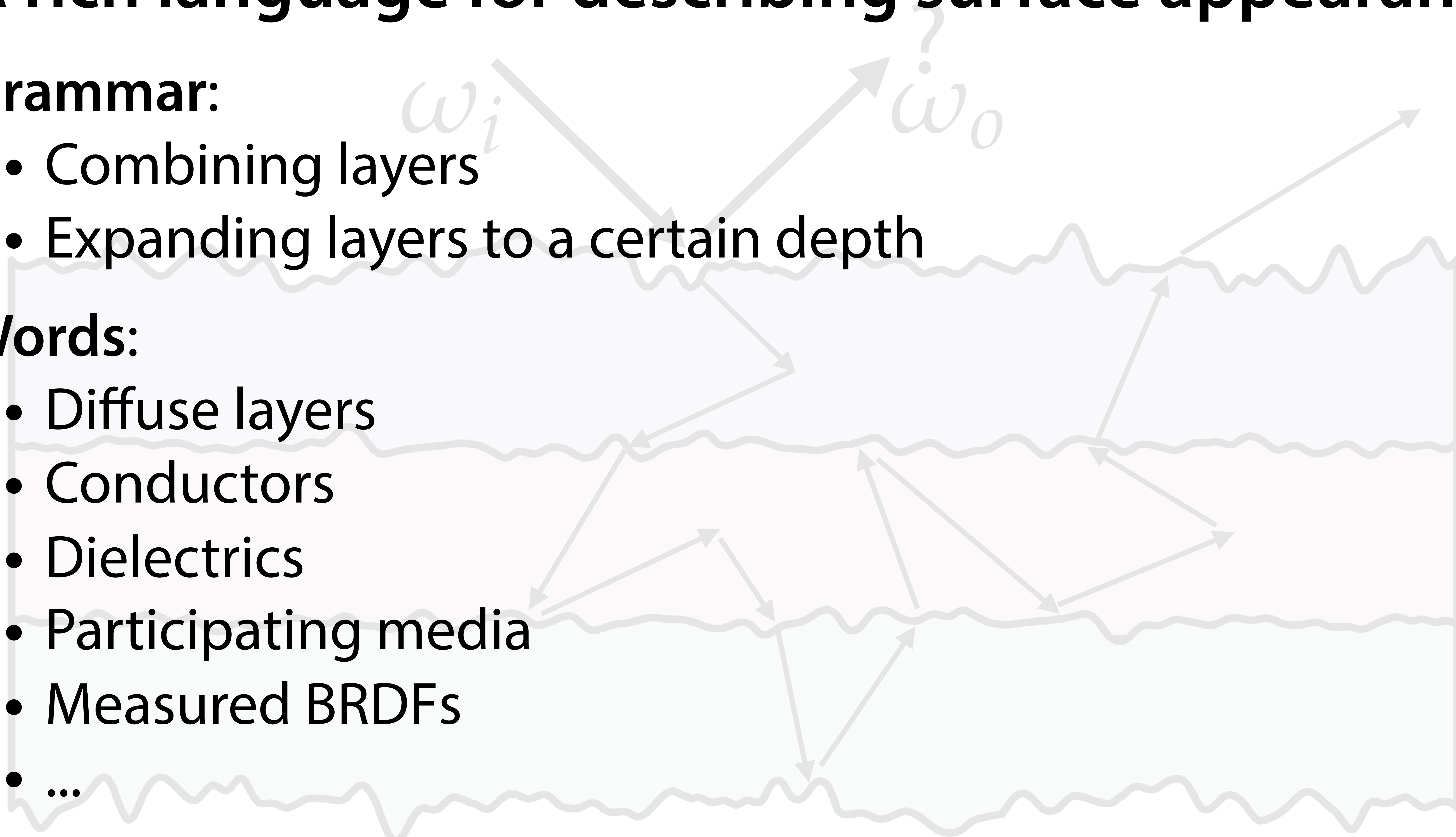
A rich language for describing surface appearance

Grammar:

- Combining layers
- Expanding layers to a certain depth

Words:

- Diffuse layers
- Conductors
- Dielectrics
- Participating media
- Measured BRDFs
- ...



A Comprehensive Framework for Rendering Layered Materials

Wenzel Jakob — ETH Zürich
Eugene D'Eon — Weta Digital
Otto Jakob — Atelier Otto Jakob
Steve Marschner — Cornell University

Code available at:

https://github.com/wjakob/layerlab

(linked in course notes)

```
In [3]: help(ll.quad)
```

```
Help on module layerlab.quad in layerlab:

NAME
    layerlab.quad - Functions for numerical quadrature

FUNCTIONS
    compositeSimpson(...) method of builtins.PyCapsule instance
        Signature : (int32_t) -> (array, array)

        Computes the nodes and weights of a composite Simpson quadrature rule with
        the given number of evaluations.

        Integration is over the interval $[-1, 1]$, which will be split into $(n-1)
        / 2$ sub-intervals with overlapping endpoints. A 3-point Simpson rule is
        applied per interval, which is exact for polynomials of degree three or
        less.

        Parameter ``n``:
            Desired number of evalution points. Must be an odd number bigger than
            3.

        Parameter ``nodes``:
            Length-``n`` array used to store the nodes of the quadrature rule

        Parameter ``nodes``:
            Length-``n`` array used to store the weights of the quadrature rule

        Remark:
            In the Python API, the ``nodes`` and ``weights`` field are returned as
            a tuple

    compositeSimpson38(...) method of builtins.PyCapsule instance
        Signature : (int32_t) -> (array, array)

        Computes the nodes and weights of a composite Simpson 3/8 quadrature rule
        with the given number of evaluations.
```

The local illumination integral

$$L_o(\omega) = \int_S f_s(\omega, \omega')\, L_i(\omega')\, |n \cdot \omega'|\, \mathrm{d}\omega'$$

The local illumination integral

outgoing radiance — BSDF — incident radiance — cosine factor

$$\underbrace{L_o(\omega)}_{\text{outgoing radiance}} = \int_{\mathcal{S}} \underbrace{f_s(\omega, \omega')}_{\text{BSDF}} \, \underbrace{L_i(\omega')}_{\text{incident radiance}} \, \underbrace{|n \cdot \omega'|}_{\text{cosine factor}} \, \mathrm{d}\omega'$$

The local illumination integral

outgoing radiance

BSDF

incident radiance

cosine factor

$$L_o(\omega) = \int_S f_s(\omega, \omega')\, L_i(\omega')\, |n \cdot \omega'|\, \mathrm{d}\omega'$$

The local illumination integral

outgoing radiance — BSDF — incident radiance — cosine factor

$$L_o(\omega) = \int_{\mathcal{S}} f_s(\omega, \omega')\, L_i(\omega')\, |n \cdot \omega'|\, \mathrm{d}\omega'$$

Write in spherical coordinates

$$L_o(\theta, \phi) = \int_0^{2\pi} \int_0^{\pi} f_s(\theta, \phi, \theta', \phi')\, L_i(\theta', \phi')\, |\cos\theta'|\, \sin\theta'\, \mathrm{d}\theta'\, \mathrm{d}\phi'$$

The local illumination integral

outgoing radiance — BSDF — incident radiance — cosine factor

$$L_o(\omega) = \int_{\mathcal{S}} f_s(\omega, \omega')\, L_i(\omega')\, |n \cdot \omega'|\, \mathrm{d}\omega'$$

Write in spherical coordinates

$$L_o(\theta, \phi) = \int_0^{2\pi} \int_0^{\pi} f_s(\theta, \phi, \theta', \phi')\, L_i(\theta', \phi')\, |\cos\theta'|\, \boxed{\sin\theta'}\, \mathrm{d}\theta'\, \mathrm{d}\phi'$$

The local illumination integral (flatland)

$$L_o(\theta, \phi) = \int_0^{2\pi} \int_0^{\pi} f_s(\theta, \phi, \theta', \phi')$$

$$L_i(\theta', \phi') \, |\cos \theta'| \, \sin \theta' \, \mathrm{d}\theta' \, \mathrm{d}\phi'$$

The local illumination integral (flatland)

$$L_o(\theta,\phi) = \int_0^{2\pi}\int_0^{\pi} f_s(\theta,\phi,\theta',\phi')$$

$$L_i(\theta',\phi')\,|\cos\theta'|\,\sin\theta'\,\mathrm{d}\theta'\,\mathrm{d}\phi'$$

Get rid of ϕ

$$L_o(\theta) = \int_0^{\pi} f_s(\theta,\theta')\,L_i(\theta')\,|\cos\theta'|\,\sin\theta'\,\mathrm{d}\theta'$$

The local illumination integral (flatland)

$$L_o(\theta) = \int_0^{\pi} f_s(\theta, \theta')\, L_i(\theta')\, |\cos\theta'|\, \sin\theta'\, \mathrm{d}\theta'$$

The local illumination integral (flatland)

$$L_o(\theta) = \int_0^{\pi} f_s(\theta, \theta')\, L_i(\theta')\, |\cos\theta'|\ \sin\theta'\, \mathrm{d}\theta'$$

Switch to $\mu = \cos\theta$

$$L_o(\mu) = \int_{-1}^{1} f_s(\mu, \mu')\, L_i(\mu')\, |\mu'|\, \mathrm{d}\mu'$$

The local illumination integral (flatland)

$$L_o(\mu) = \int_{-1}^{1} f_s(\mu, \mu')\, L_i(\mu')\, |\mu'|\, \mathrm{d}\mu'$$

The local illumination integral (flatland)

$$L_o(\mu) = \int_{-1}^{1} f_s(\mu, \mu')\, L_i(\mu')\, |\mu'|\, \mathrm{d}\mu'$$

Discretize for $\mu_1, \ldots, \mu_n$

$$L_o(\mu_j) = \sum_{i=1}^{n} f_s(\mu_j, \mu_i)\, |\mu_i|\; w_i\, L_i(\mu_i)$$

The local illumination integral (flatland)

$$L_o(\mu) = \int_{-1}^{1} f_s(\mu, \mu')\, L_i(\mu')\, |\mu'|\, \mathrm{d}\mu'$$

Discretize for $\mu_1, \ldots, \mu_n$

$$L_o(\mu_j) = \sum_{i=1}^{n} f_s(\mu_j, \mu_i)\, |\mu_i|\; w_i\, L_i(\mu_i)$$

Write using matrix notation

$$\mathbf{L}_o = \mathbf{F}_s \mathbf{L}_i$$

The local illumination integral (flatland)

The local illumination integral (flatland)

$$\mathbf{L}_o = \mathbf{F}_s \mathbf{L}_i$$

The local illumination integral (flatland)

$$\mathbf{L}_o = \boxed{\mathbf{F}_s} \mathbf{L}_i$$

"Scattering matrix"

Instantiating and plotting layers

In [3]:
```
mu, w = ll.quad.gaussLobatto(10)
mu, w
```

Out[3]:
```
(array([-1.        , -0.91953391, -0.73877387, -0.47792495, -0.16527896,
         0.16527896,  0.47792495,  0.73877387,  0.91953391,  1.        ]),
 array([ 0.02222222,  0.13330599,  0.22488934,  0.29204268,  0.32753976,
         0.32753976,  0.29204268,  0.22488934,  0.13330599,  0.02222222]))
```

Instantiating and plotting layers

In [3]:
```
mu, w = ll.quad.gaussLobatto(10)
mu, w
```

Out[3]:
```
(array([-1.        , -0.91953391, -0.73877387, -0.47792495, -0.16527896,
         0.16527896,  0.47792495,  0.73877387,  0.91953391,  1.        ]),
 array([ 0.02222222,  0.13330599,  0.22488934,  0.29204268,  0.32753976,
         0.32753976,  0.29204268,  0.22488934,  0.13330599,  0.02222222]))
```

In [4]:
```
def plot_layer(layer, num_samples = 200, phi_d = 0):
    fig = plt.figure(tight_layout = True)
    titles = [ '$T^{bt}$', '$R^{t}$', '$R^{b}$', '$T^{tb}$' ]
    for i in range(4):
        # Plot extents for subplot
        extent = [i%2-1, i%2, i//2-1, i//2]
        # Initialize points where the layer is evaluated
        mu_i = np.linspace(extent[0], extent[1], num_samples)
        mu_o = np.linspace(extent[2], extent[3], num_samples)
        mu_i_arg, mu_o_arg = np.meshgrid(mu_i, mu_o)
        # Evaluate the layer scattering function
        result = layer.eval(mu_o_arg, mu_i_arg, phi_d)
        # Scale by cosine factors to plot scattered energy
        result = np.array(result) * np.abs(mu_i_arg * mu_o_arg)
        # Plot result
        fig.add_subplot(2, 2, i + 1)
        plt.title(titles[i])
        plt.imshow(result, extent = extent, aspect = 'equal', origin='lower',
                   vmin = 0, vmax = np.percentile(result, 99))
        plt.gca().invert_yaxis()
        plt.colorbar()
    plt.show()
```

Instantiating and plotting layers

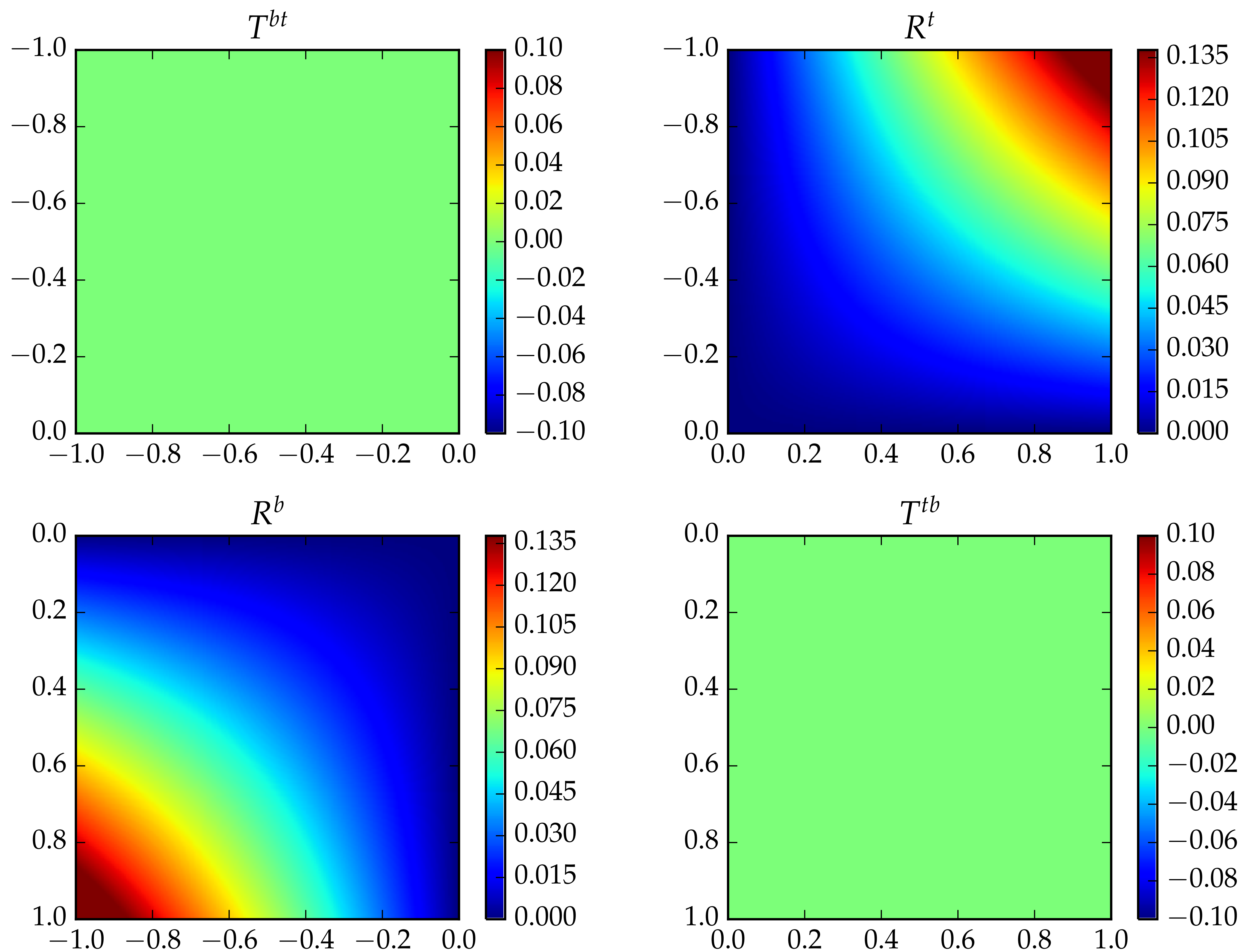
In [5]:

```
layer = ll.Layer(mu, w)
layer.setDiffuse(albedo = 0.5)

plot_layer(layer)
```

Instantiating and plotting layers

In [5]:

```
layer = ll.Layer(mu, w)
layer.setDiffuse(albedo = 0.5)

plot_layer(layer)
```

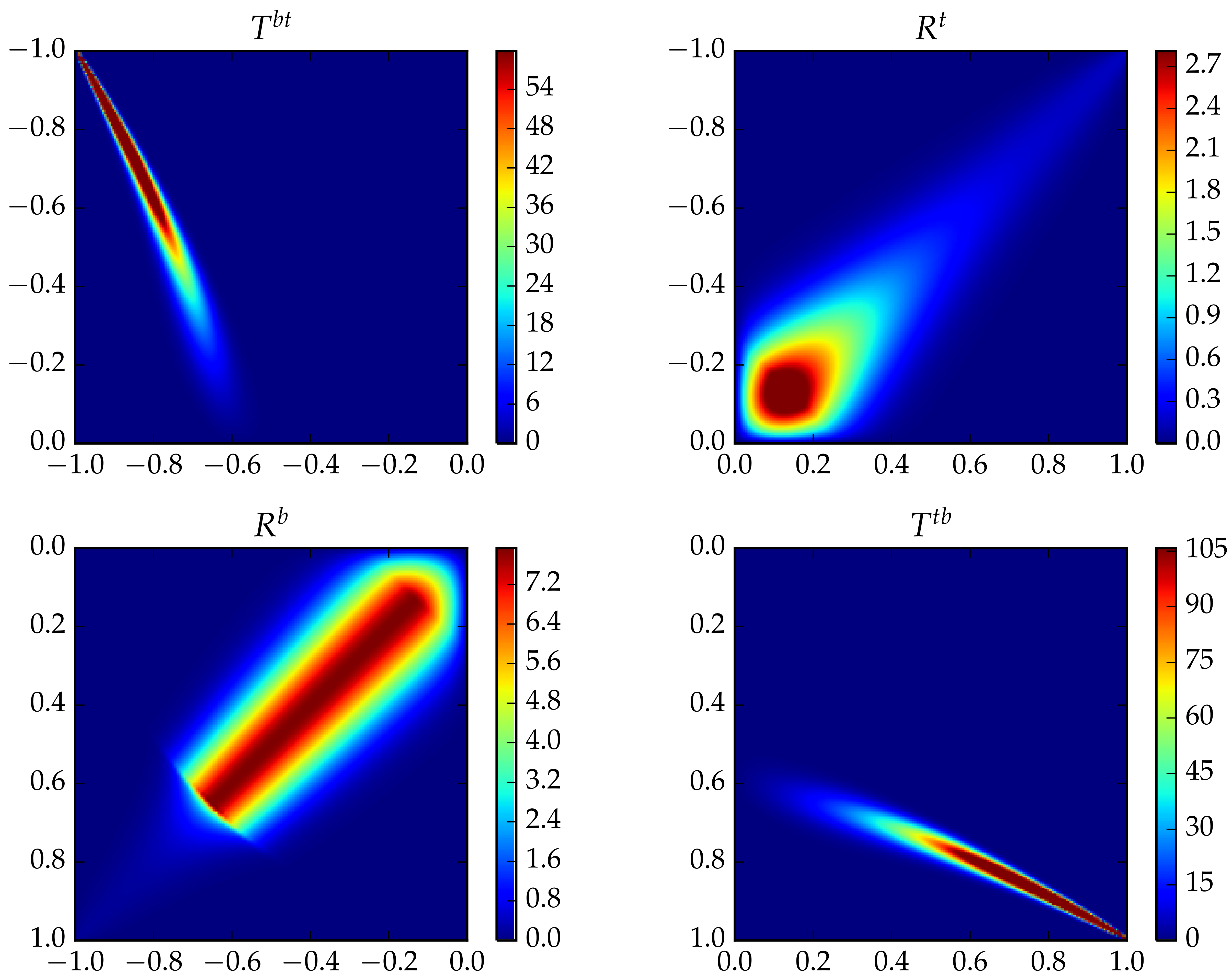


Instantiating and plotting layers

In [6]:
```
mu, w = ll.quad.gaussLobatto(200)
dielectric = ll.Layer(mu, w, 200)
dielectric.setMicrofacet(eta = 1.5, alpha = 0.1)
plot_layer(dielectric)
```

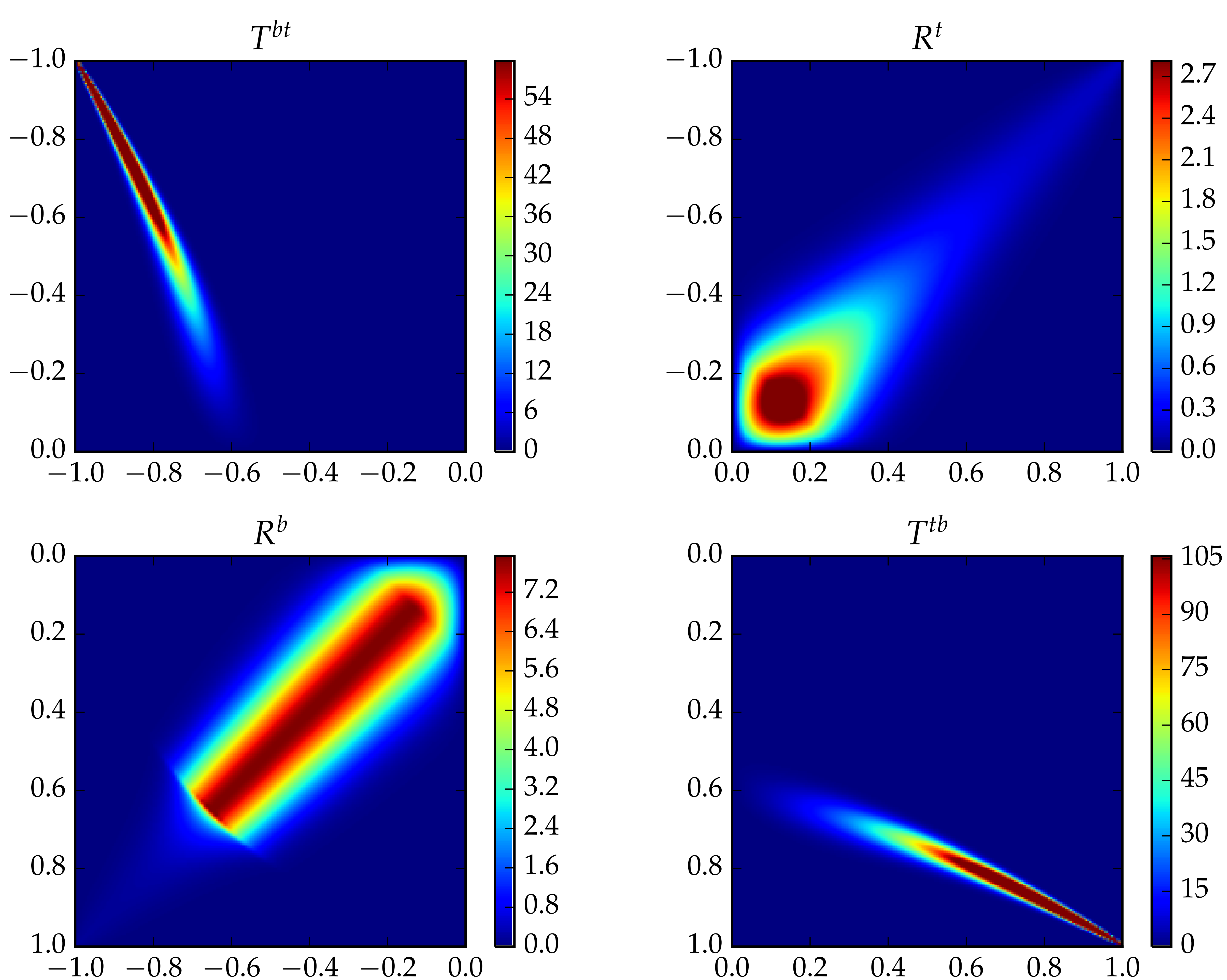
Instantiating and plotting layers

In [6]:
```
mu, w = ll.quad.gaussLobatto(200)
dielectric = ll.Layer(mu, w, 200)
dielectric.setMicrofacet(eta = 1.5, alpha = 0.1)
plot_layer(dielectric)
```

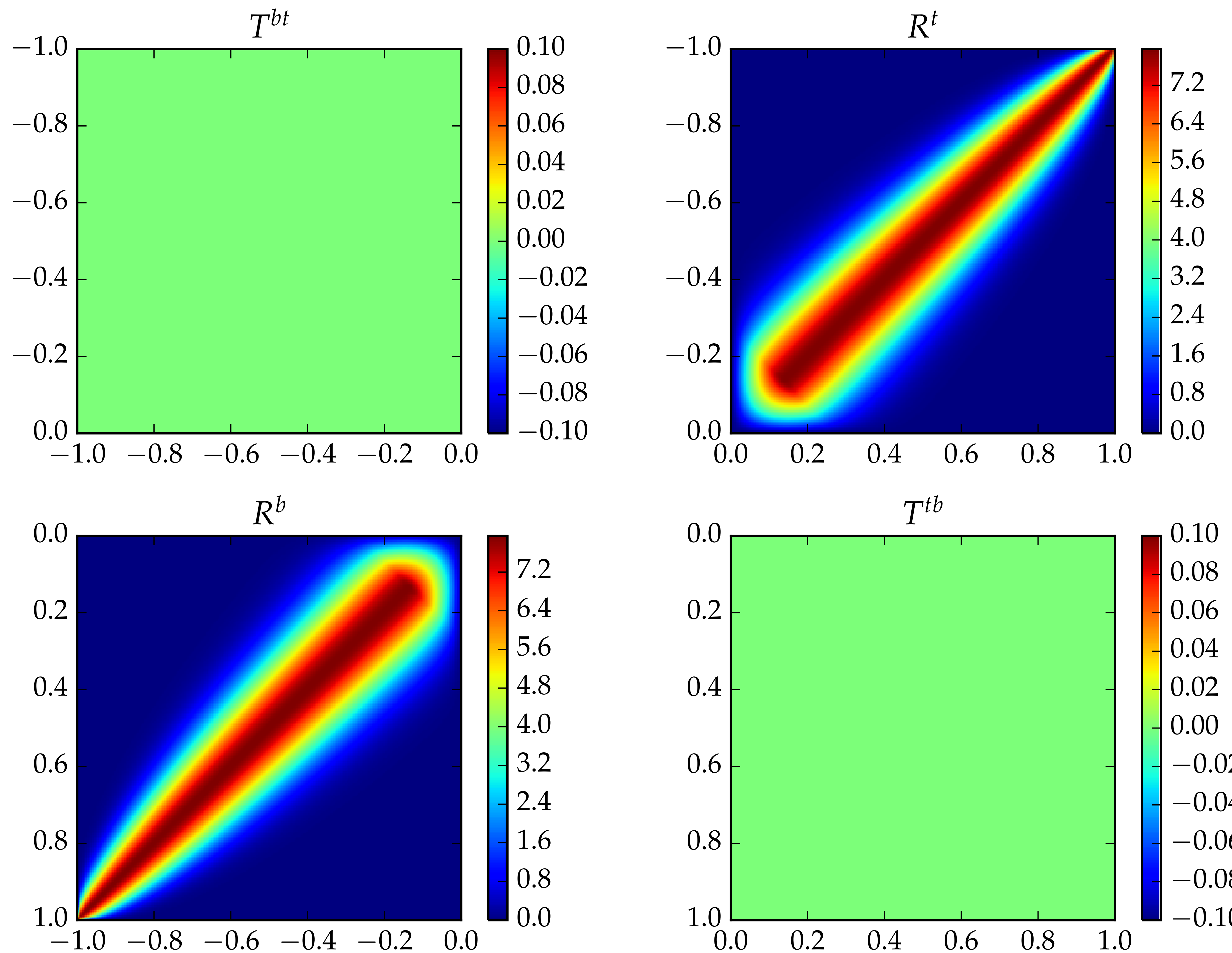


Instantiating and plotting layers

In [6]:
```python
mu, w = ll.quad.gaussLobatto(200)
dielectric = ll.Layer(mu, w, 200)
dielectric.setMicrofacet(eta = 1.5, alpha = 0.1)
plot_layer(dielectric)
```



In [7]:
```python
mu, w = ll.quad.gaussLobatto(200)
conductor = ll.Layer(mu, w, 200)
conductor.setMicrofacet(eta = 0+1j, alpha = 0.1)
plot_layer(conductor)
```



Outline

- What is layerlab?
- Benefits of layered material models
- Layers in flatland
- **Combining layers**
- Layers in 3D
- Experiments

Glass Plates Theory [Stokes 1860]

Air

Glass

Air

Glass Plates Theory [Stokes 1860]

Glass Plates Theory [Stokes 1860]

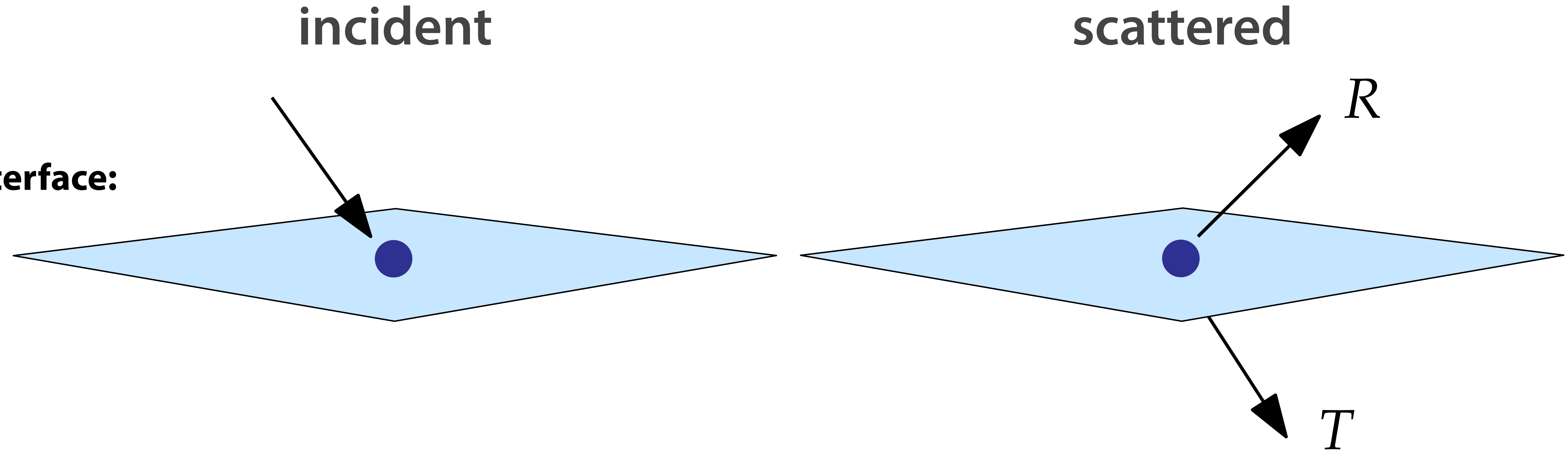
Glass Plates Theory [Stokes 1860]

Glass Plates Theory [Stokes 1860]

$$R_{\text{tot}} = R + TRT + \ \ldots \qquad = R + T^2 \sum_{i=0}^{\infty} R^{2i+1} \quad = R + \frac{RT^2}{1-R^2},$$

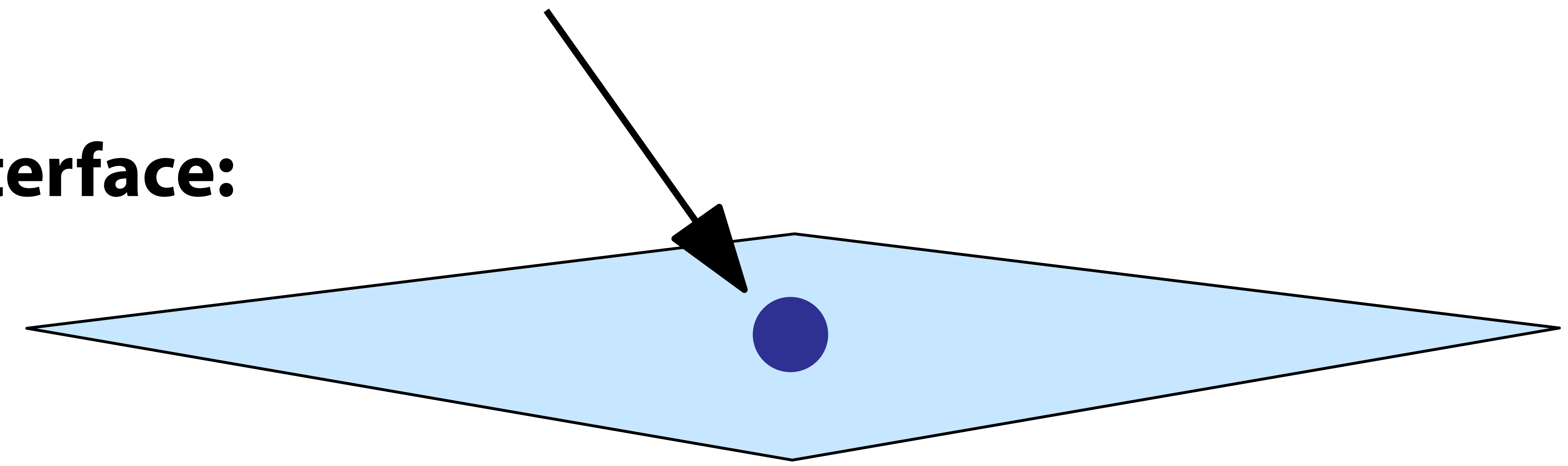
$$T_{\text{tot}} = TT + TR^2T + \ \ldots \quad = T^2 \sum_{i=0}^{\infty} R^{2i} \qquad = \frac{T^2}{1-R^2}.$$

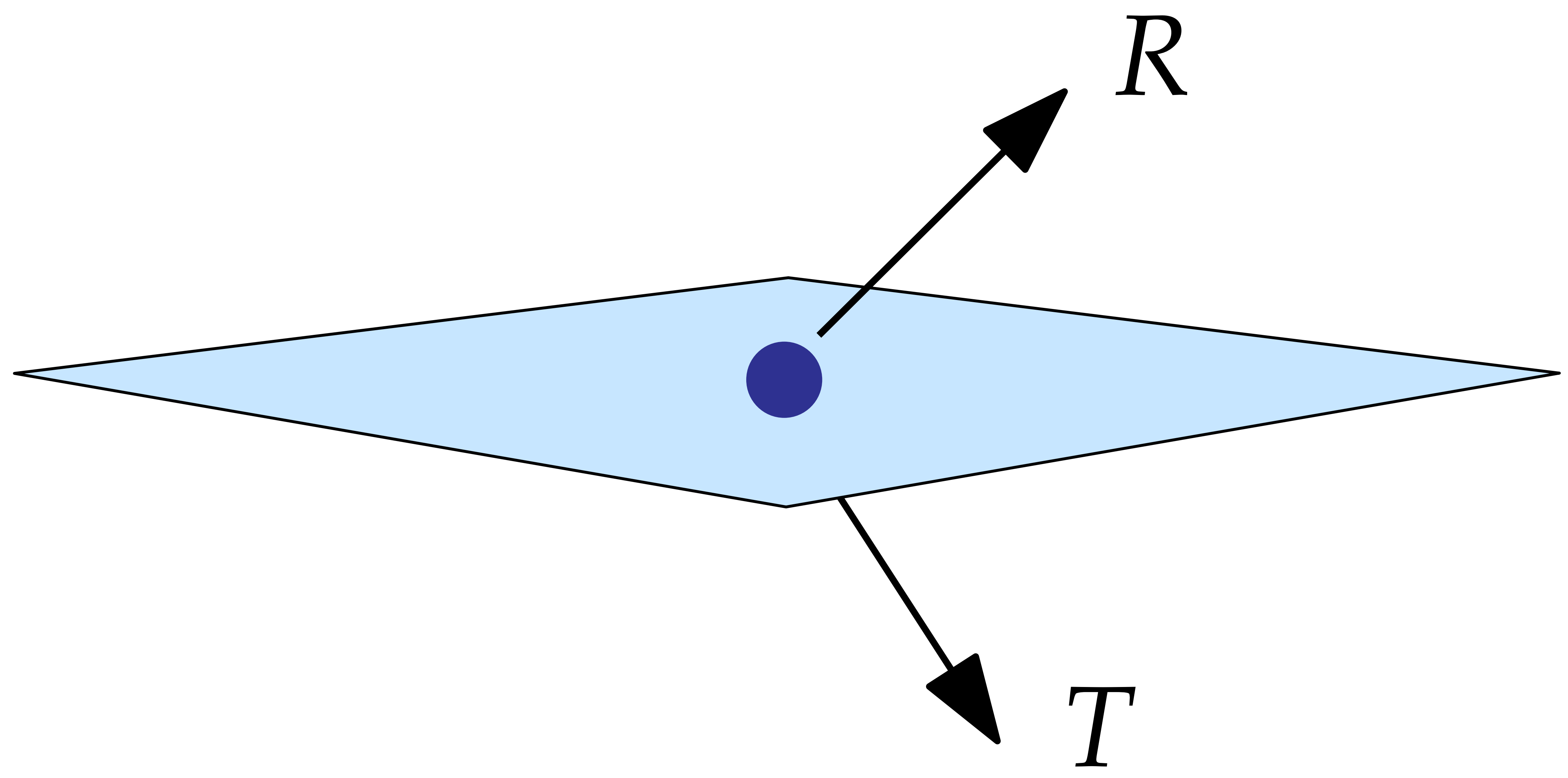
Glass interface:



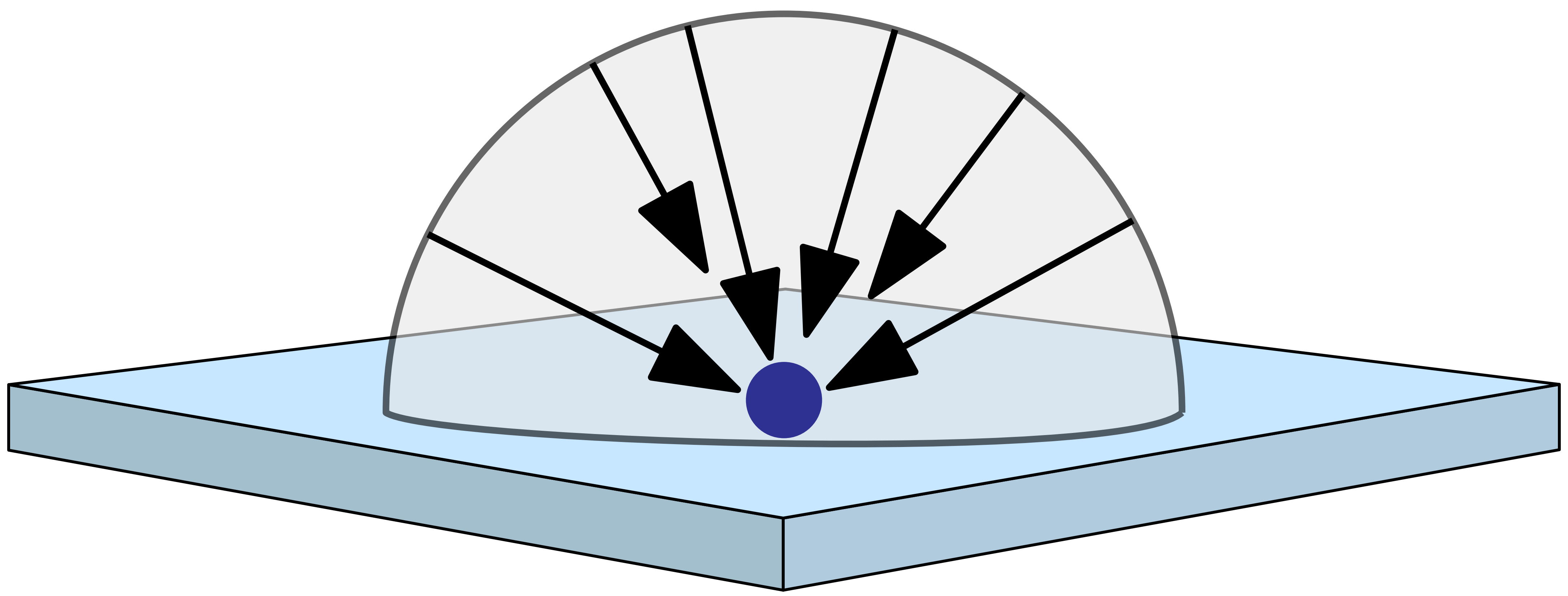
incident | scattered

Glass interface:



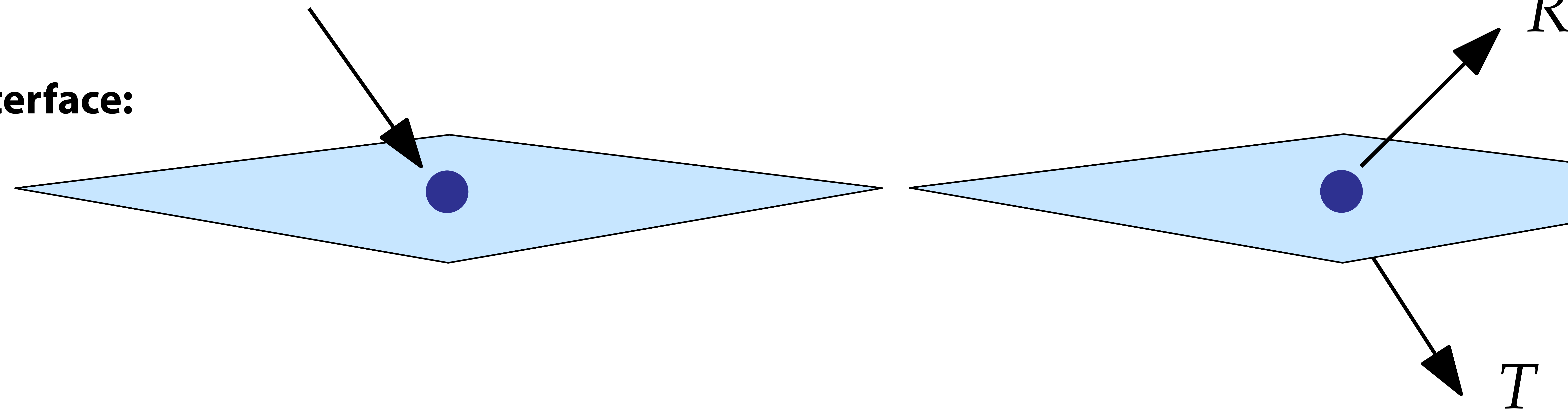


General:

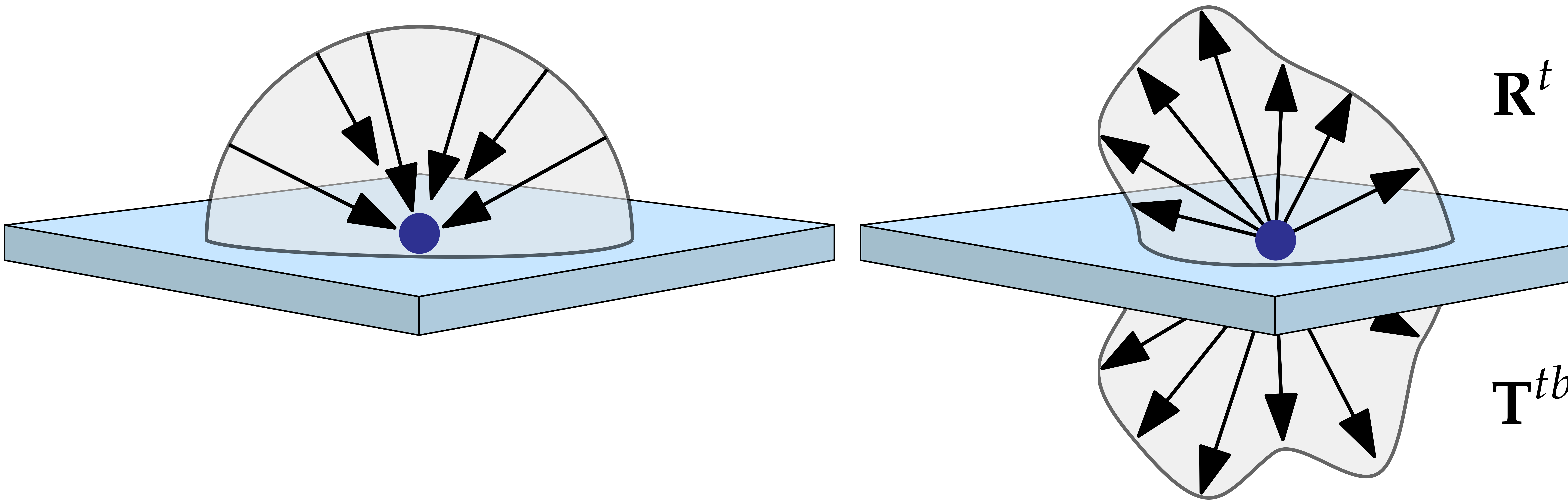


incident **scattered**

Glass interface:

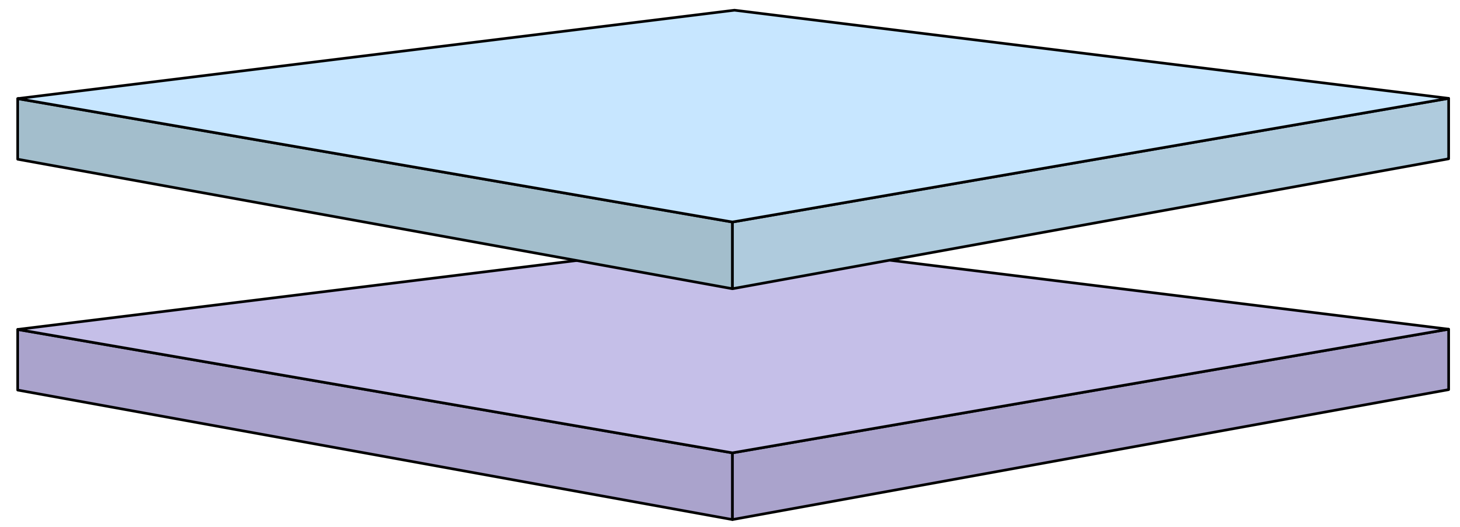


General:



t: top, b: bottom

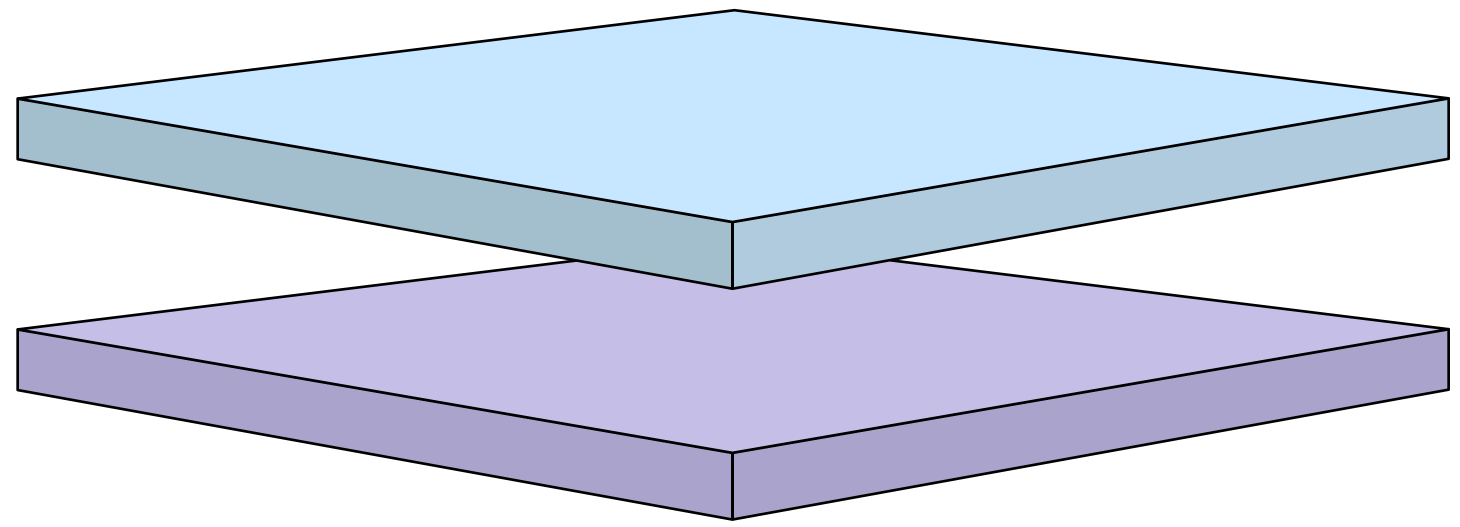
Adding equations



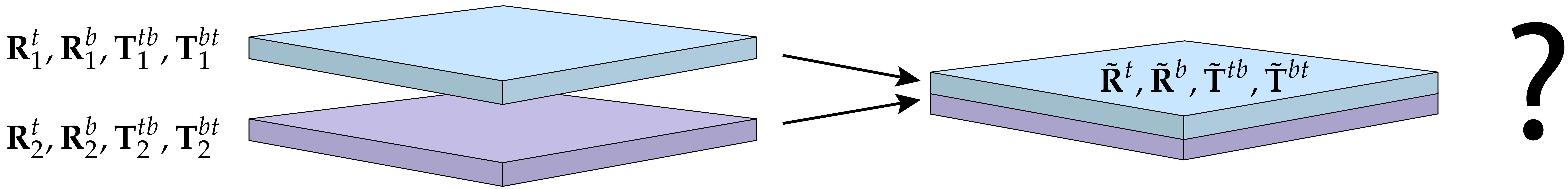
Adding equations

$\mathbf{R}_1^t, \mathbf{R}_1^b, \mathbf{T}_1^{tb}, \mathbf{T}_1^{bt}$

$\mathbf{R}_2^t, \mathbf{R}_2^b, \mathbf{T}_2^{tb}, \mathbf{T}_2^{bt}$



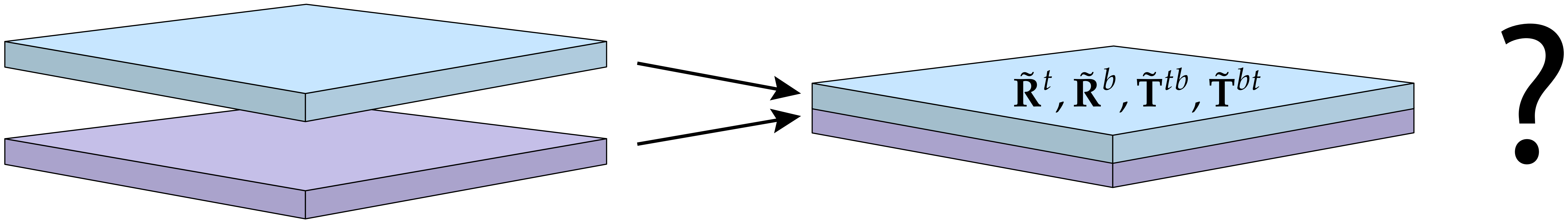
Adding equations



Adding equations

$\mathbf{R}_1^t, \mathbf{R}_1^b, \mathbf{T}_1^{tb}, \mathbf{T}_1^{bt}$

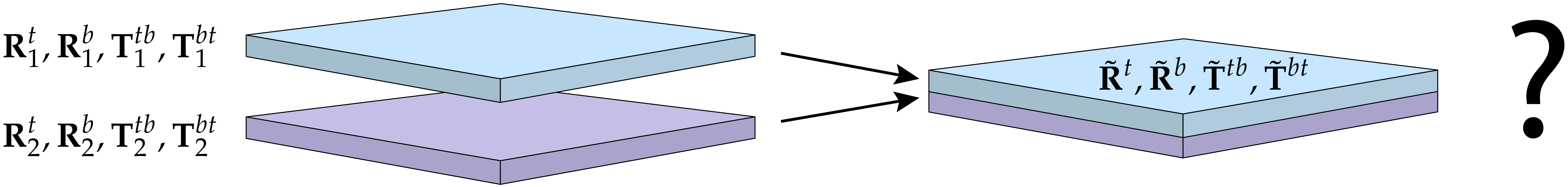
$\mathbf{R}_2^t, \mathbf{R}_2^b, \mathbf{T}_2^{tb}, \mathbf{T}_2^{bt}$



$$
\boxed{
\begin{aligned}
\tilde{\mathbf{R}}^t &= \mathbf{R}_1^t + \mathbf{T}_1^{bt}(\mathbf{I} - \mathbf{R}_2^t\mathbf{R}_1^b)^{-1}\mathbf{R}_2^t\mathbf{T}_1^{tb} \\
\tilde{\mathbf{R}}^b &= \mathbf{R}_2^b + \mathbf{T}_2^{tb}(\mathbf{I} - \mathbf{R}_1^b\mathbf{R}_2^t)^{-1}\mathbf{R}_1^b\mathbf{T}_2^{bt} \\
\tilde{\mathbf{T}}^{tb} &= \mathbf{T}_2^{tb}(\mathbf{I} - \mathbf{R}_1^b\mathbf{R}_2^t)^{-1}\mathbf{T}_1^{tb} \\
\tilde{\mathbf{T}}^{bt} &= \mathbf{T}_1^{bt}(\mathbf{I} - \mathbf{R}_2^t\mathbf{R}_1^b)^{-1}\mathbf{T}_2^{bt}
\end{aligned}
}
$$

[Grant and Hunt 1969]

Adding equations



$$\boxed{\begin{aligned}
\tilde{\mathbf{R}}^t &= \mathbf{R}_1^t + \mathbf{T}_1^{bt}(\mathbf{I} - \mathbf{R}_2^t\mathbf{R}_1^b)^{-1}\mathbf{R}_2^t\mathbf{T}_1^{tb} \\
\tilde{\mathbf{R}}^b &= \mathbf{R}_2^b + \mathbf{T}_2^{tb}(\mathbf{I} - \mathbf{R}_1^b\mathbf{R}_2^t)^{-1}\mathbf{R}_1^b\mathbf{T}_2^{bt} \\
\tilde{\mathbf{T}}^{tb} &= \mathbf{T}_2^{tb}(\mathbf{I} - \mathbf{R}_1^b\mathbf{R}_2^t)^{-1}\mathbf{T}_1^{tb} \\
\tilde{\mathbf{T}}^{bt} &= \mathbf{T}_1^{bt}(\mathbf{I} - \mathbf{R}_2^t\mathbf{R}_1^b)^{-1}\mathbf{T}_2^{bt}
\end{aligned}}$$

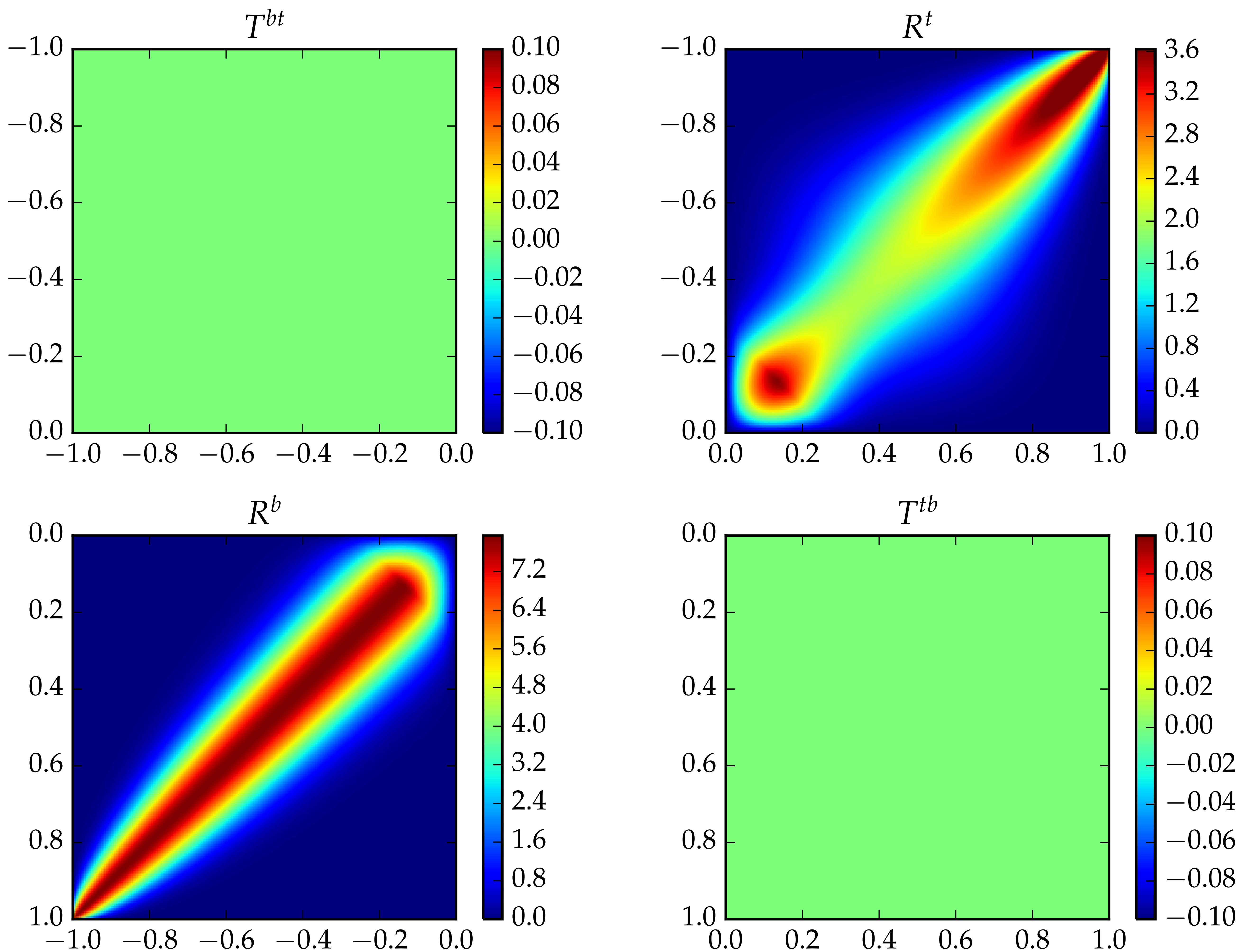
$$R + \frac{RT^2}{1 - R^2}$$

$$\frac{T^2}{1 - R^2}$$

[Grant and Hunt 1969]

Combining layers

In [8]:
```
conductor.addToTop(dielectric)
plot_layer(conductor)
```



Combining layers

In [8]:
```
conductor.addToTop(dielectric)
plot_layer(conductor)
```

Combining layers

```
In [8]: conductor.addToTop(dielectric)
        plot_layer(conductor)
```

Layers in 3D (with azimuth!)

Layers in 3D (with azimuth!)

$$\mathbf{L}_o = \mathbf{F}_s \mathbf{L}_i$$

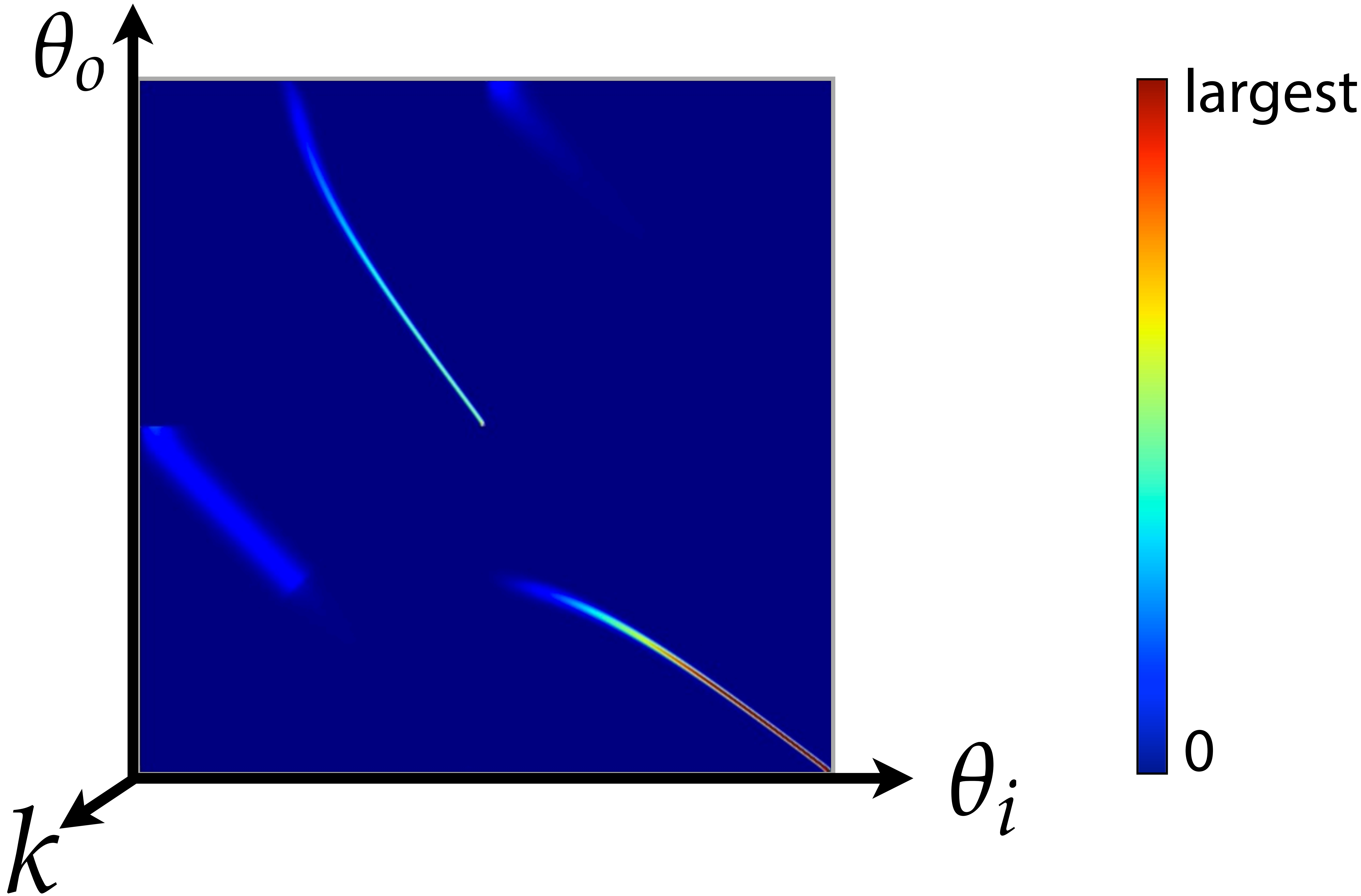
Layers in 3D (with azimuth!)

$$\mathbf{L}_o = \mathbf{F}_s \mathbf{L}_i$$

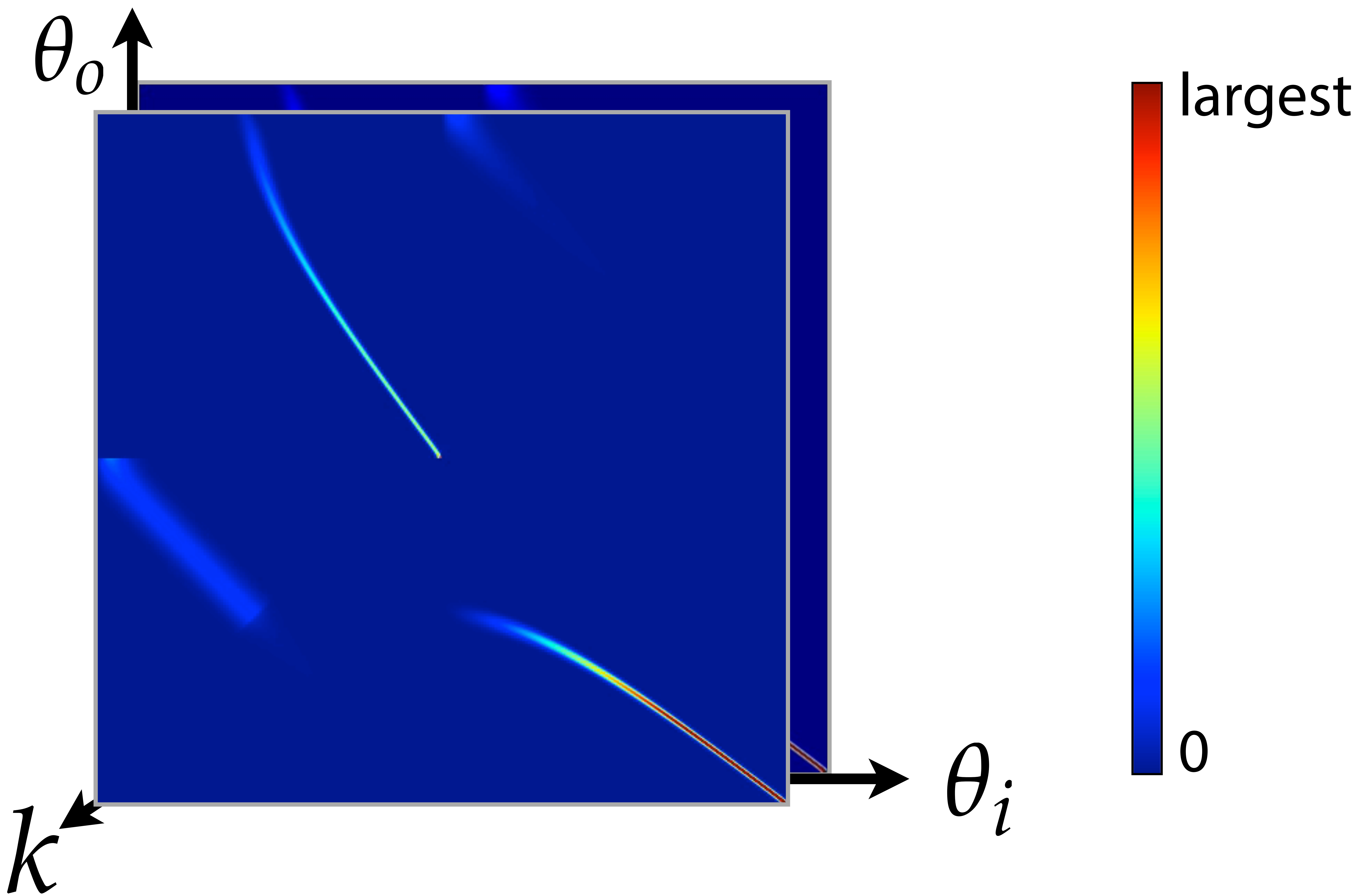
Frequency space

$$\mathbf{L}_o^{(l)} = \mathbf{F}_s^{(l)} \mathbf{L}_i^{(l)} \quad (l = 0, \ldots)$$

Sparse matrix representation



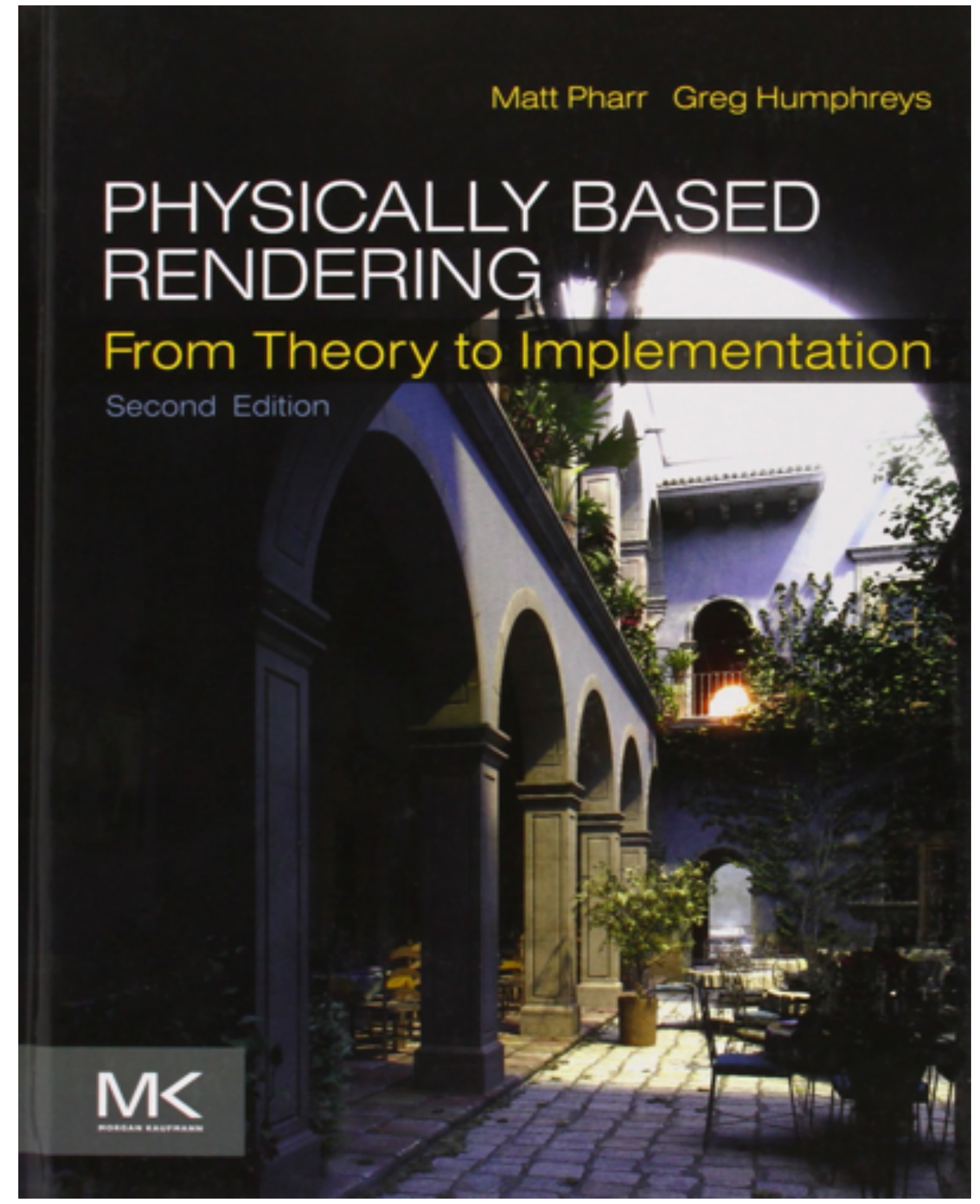
Sparse matrix representation



Writing parameters to disk

In [9]:
```
ll.BSDFStorage.fromLayer("output.bsdf", layer)
```

Out[9]:
```
BSDFStorage[
  mmap = MemoryMappedFile[filename="output.bsdf", size=3.3 MiB],
  nNodes = 202,
  nMaxOrder = 200,
  nChannels = 1,
  nBases = 1,
  eta = 1
]
```



Output can be rendered with Mitsuba and PBRT version 3
code @ https://github.com/mmp/pbrt-v3

Mitsuba
Physically Based Renderer

Outline

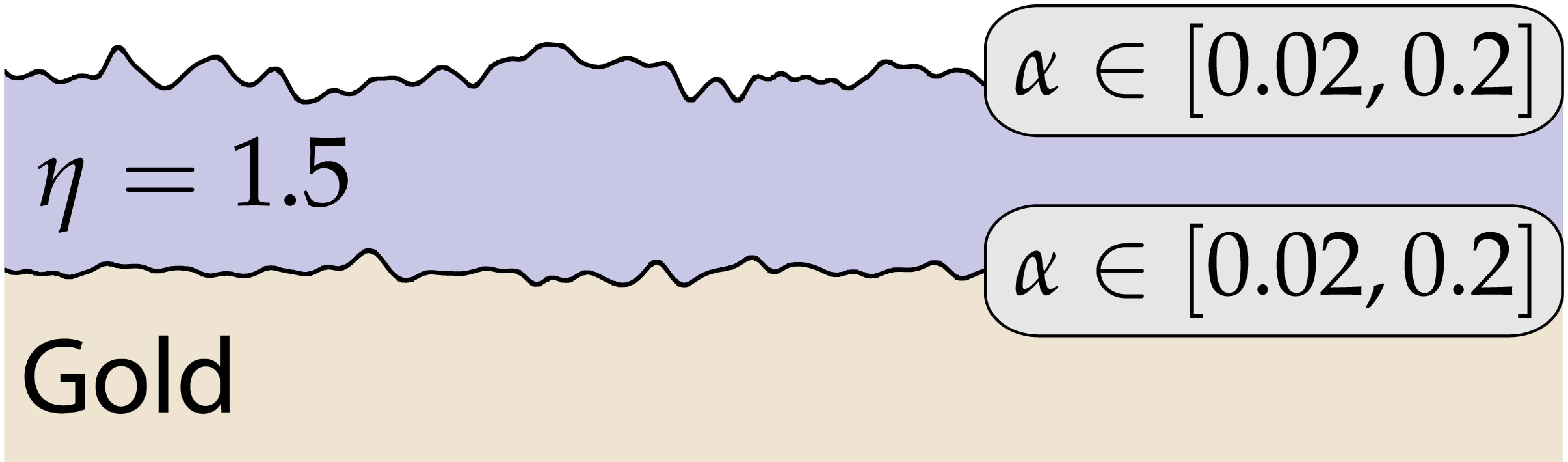
- What is layerlab?
- Benefits of layered material models
- Layers in flatland
- Combining layers
- Layers in 3D
- **Experiments**

Interaction of rough boundaries

- Stanford dragon with two layers
  - 1. Rough dielectric interface
  - 2. Rough gold interface

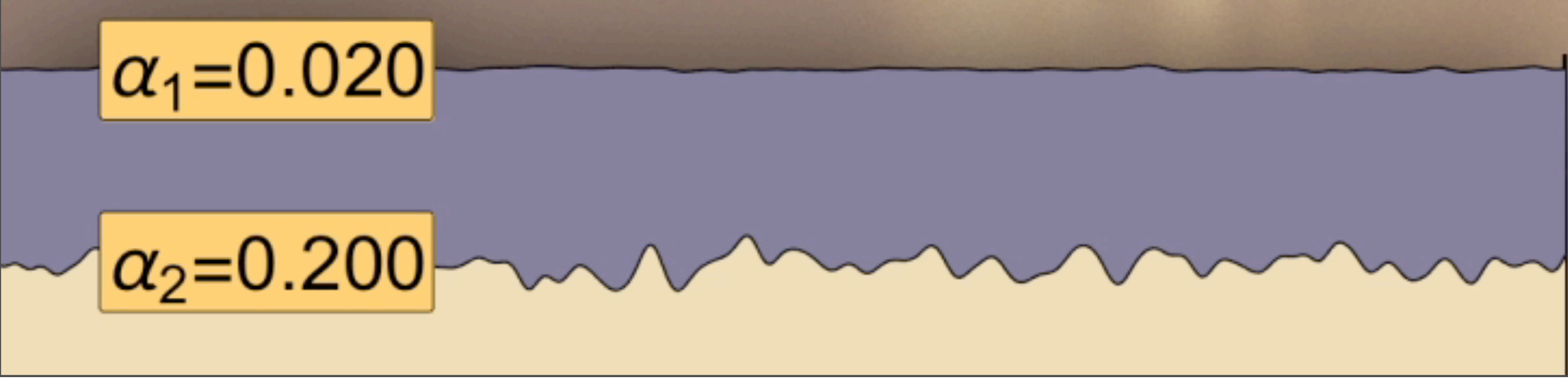
Matrix resolution $n \in [55, 1097]$

Fourier orders $m \in [92, 4194]$



Varying top layer, bottom layer smooth

Varying bottom layer, top layer smooth



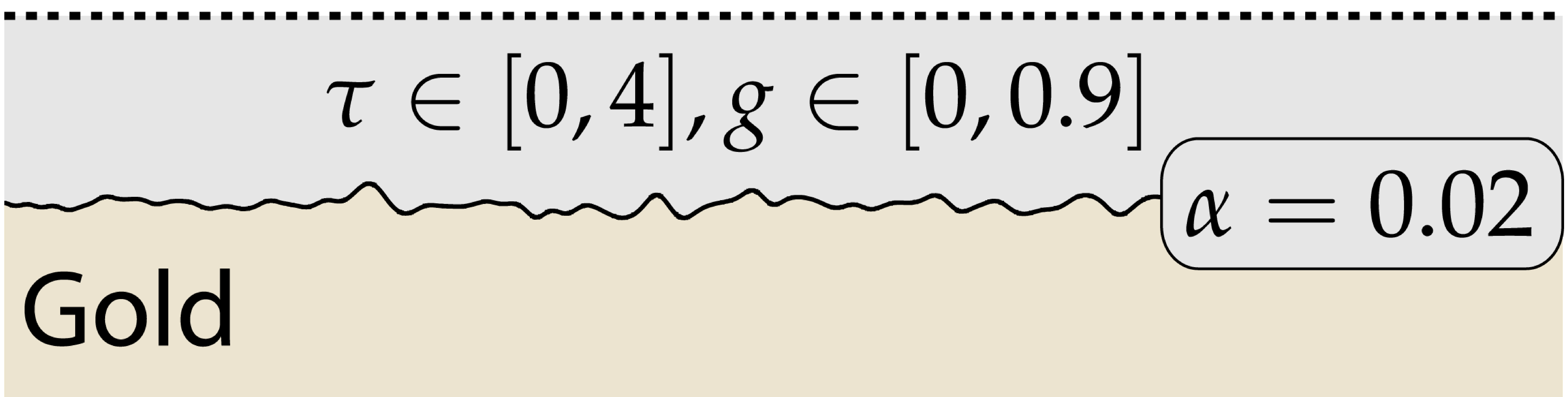

α_1=0.200
α_2=0.200

Scattering dust on metal

- Stanford dragon with two layers
  - 1. Index-matched scattering layer
  - 2. Rough gold interface

Matrix resolution $n = 4194$

Fourier orders $m = 4194$




α=0.02
τ=0.000, g=0.900

Varying layer anisotropy

Varying layer width (isotropic)

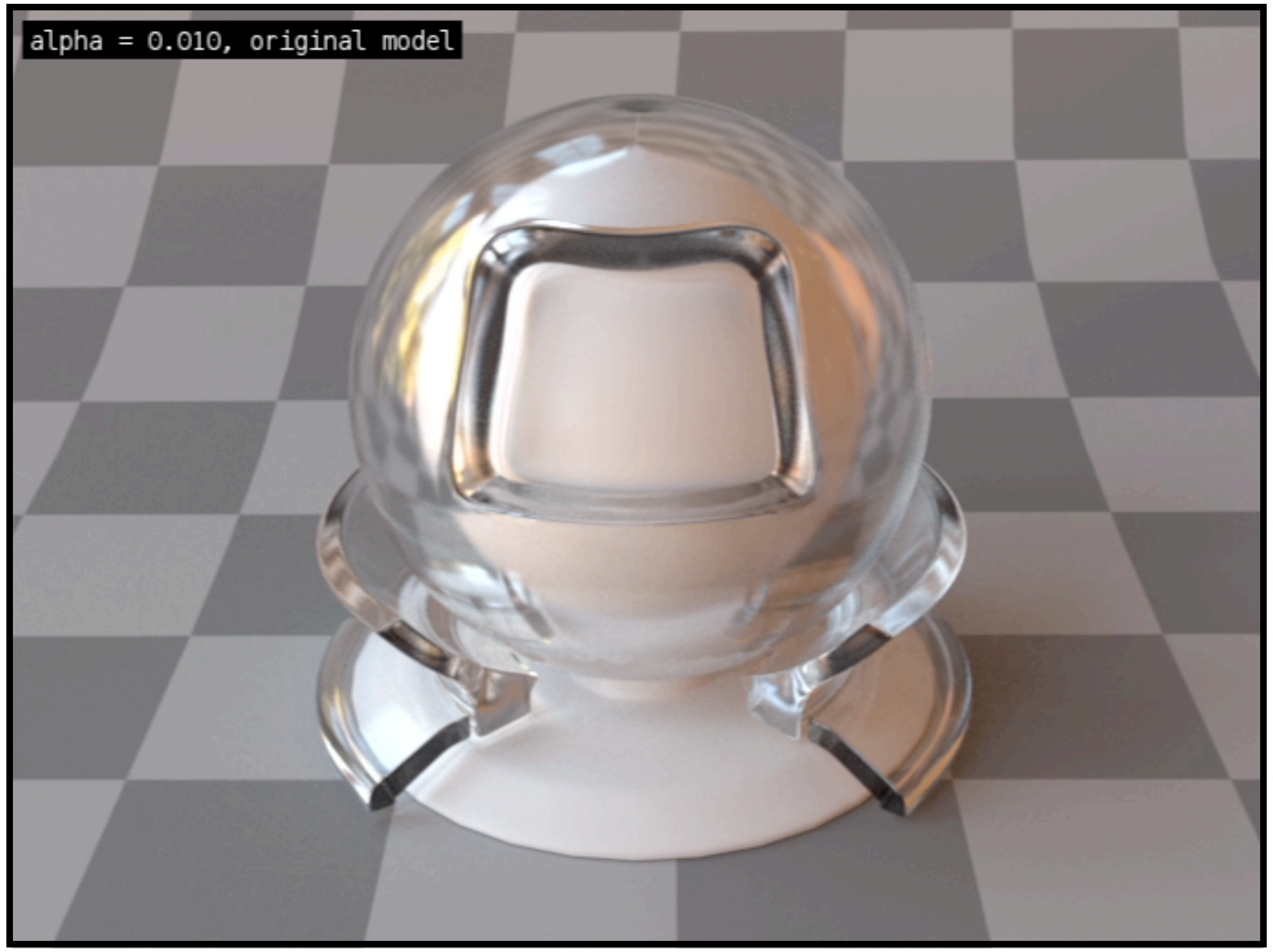
Multiple scattering correction

Multiple scattering correction

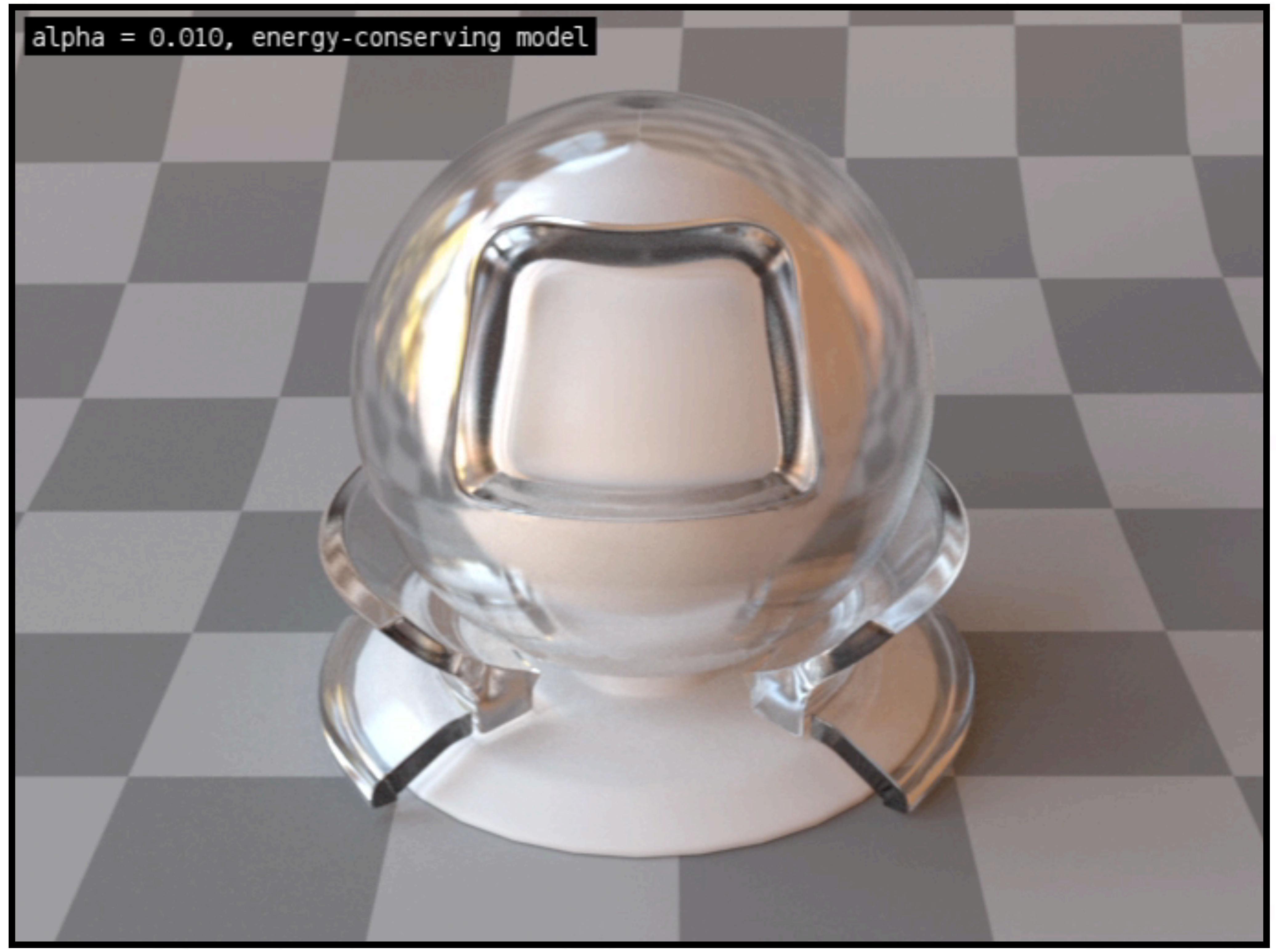
Multiple scattering correction

Multiple scattering correction

Multiple scattering term for dielectrics



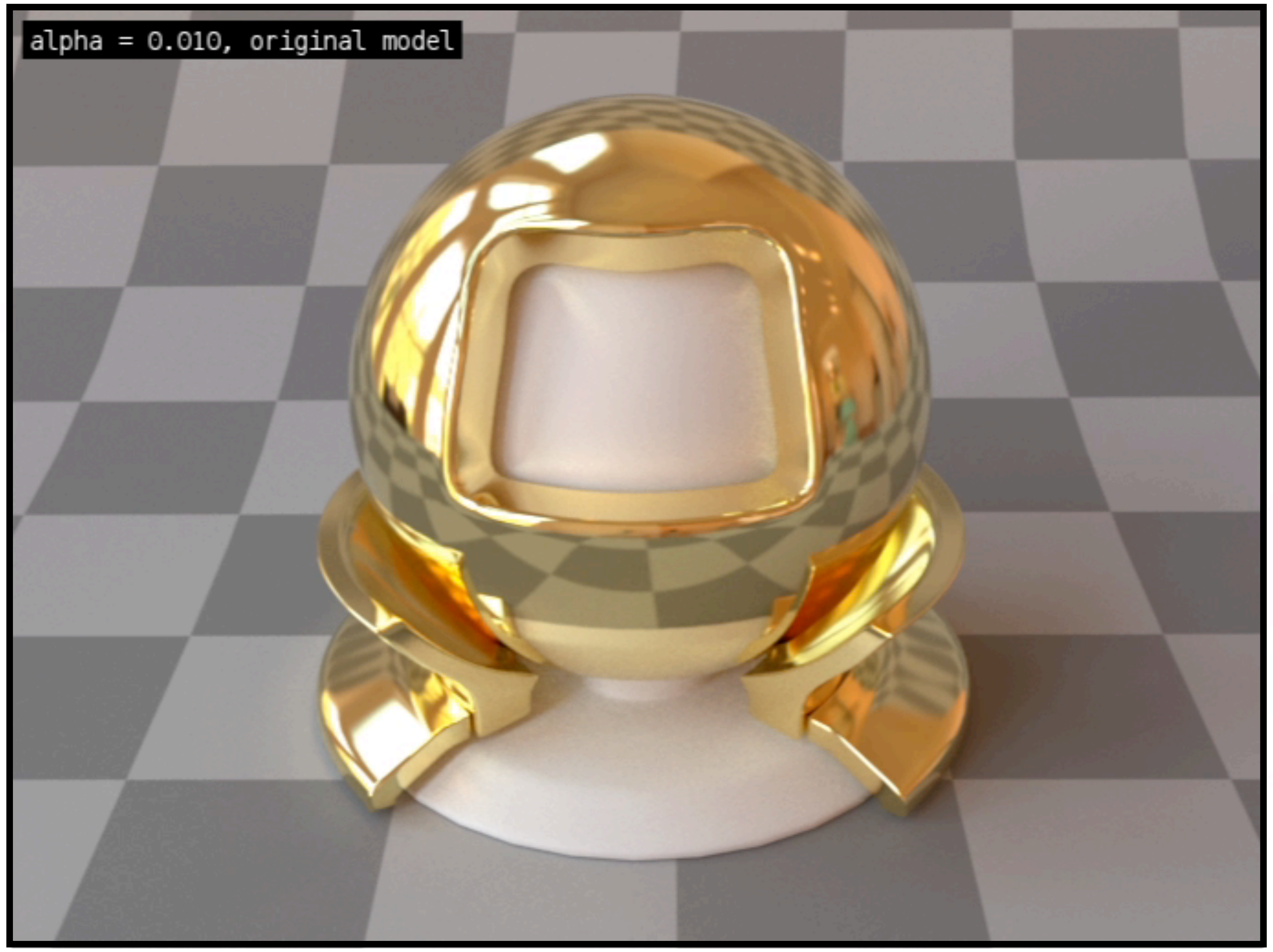
Standard microfacet model



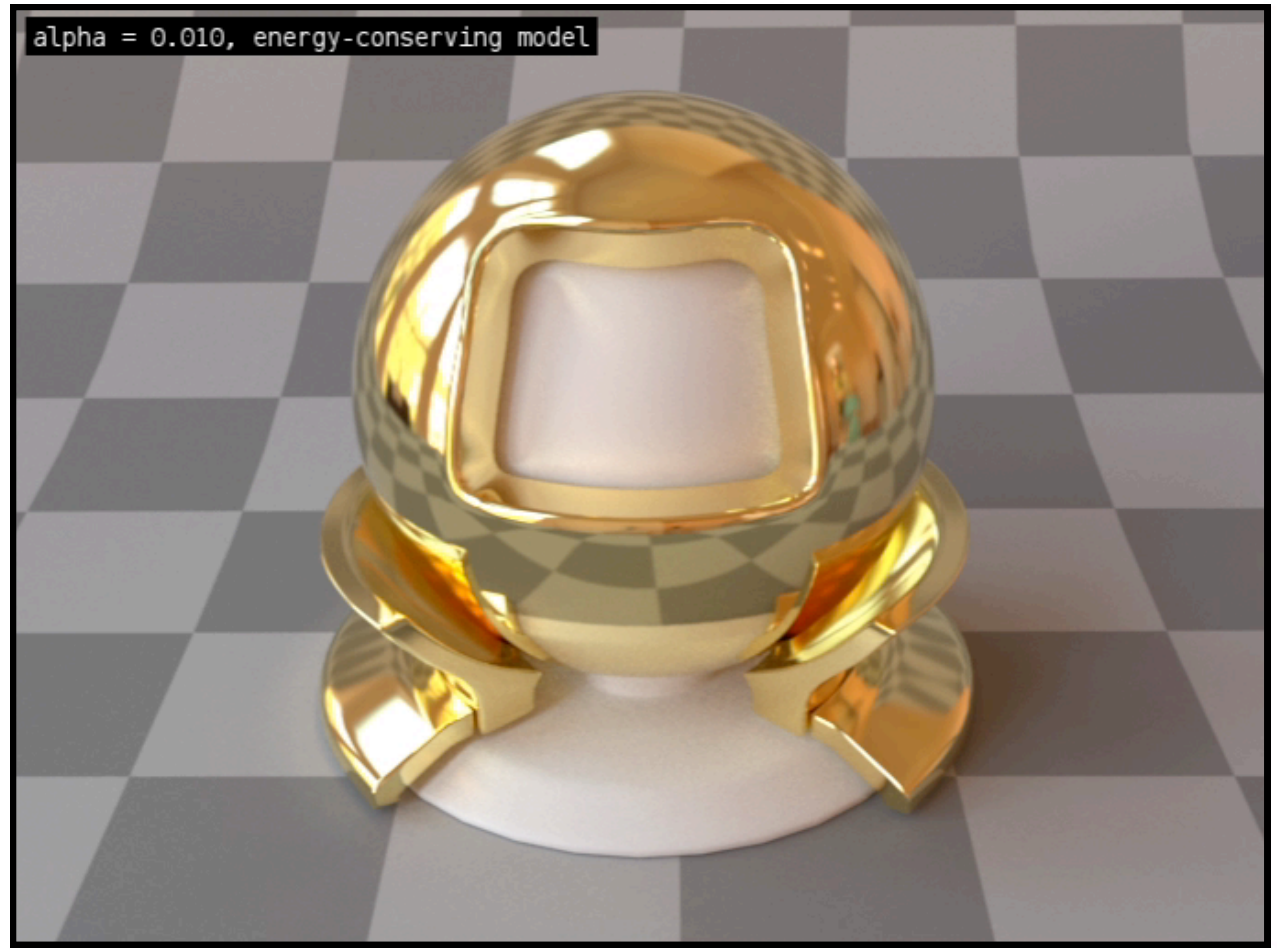
Energy conserving model

$(\alpha = 0.01, \ldots, 2)$

Multiple scattering term for conductors



Standard microfacet model



Energy conserving model

$(\alpha = 0.01, \ldots, 2)$

Summary

- Realistic toolkit for simulating layered materials
- Extremely accurate (see course notes)
- No existing rendering software does this today :(
- Remaining challenges: texturing multiple parameters.