A Guide to Variable-Resolution Geometries in MEEP C++

Felix Schwarz

felix.schwarz@tu-ilmenau.de,
Technische Universität Ilmenau,
Institut für Physik,
D-98693 Ilmenau

November 3, 2017

This gives a little overview of the procedure used to gain variable resolution using MEEP as a library in C++. This document was intended purely as a note for myself and should be read as such. I programmed a couple of examples that should be grouped together in archives with this document. The code is probably neither fast nor well written, but it served its purpose for me.

The geometry transformation is sketched in figure 1.

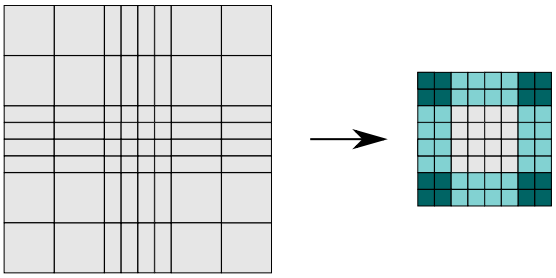

Figure 1: Sketch of the variable-resolution grid (left), which can be transformed into a standard geometry with anisotropic optical properties (right).

The coordinate stretching can be absorbed into a transformation of ε, μ, ρ, $\mathbf{J}$ and the fields in Maxwells Equations see for example S. Johnsons's notes on a MIT course, "Coordinate Transformation & Invariance in Electromagnetism".

Implementation Details

Define a grid volume as usual, using the maximal resolution of the central region. Calculate the appropriate scaled size beforehand and reduce the size of the outer regions accordingly when defining the volume.

Defined a class that handles the different resolution domains, including functions that give the scale factors for the $\chi_1 + 1$, i.e. ε or μ respectively, transformation and the $\mathbf{J}$ transformation. Note that $\chi_1 + 1$ now definitely is a (diagonal) tensor in the domains, where the stretched geometry has rectangular pixels, so the stretching factors

depend on the field component. The input for these member functions is therefore a position vector and a component direction.

Next, define a material property class. Because we want to specify tensor-valued $\chi_1 + 1$, we directly have to override the virtual `eff_chi1_inv` function. This function also handles anisotropic averaging, so this also has to be implemented. **You might be tempted to ignore the anisotropic averaging, but this is very unwise, since the resolution-domain boundaries will cause unphysical reflections if you forgo the averaging.** Cascading the resolution, i.e. use more domains in order to reduce the difference between adjacent domains, even worsens reflection in my examples (I just made one large jump). Tests show, however, that you can significantly rise the tolerances and lower the maximum iterations of this (originally fall-back) procedure without compromising the physics very much. So, for now, just copy and paste the anisotropic avergaing code into our function, together with the `normal_vector` (renamed to `normal_vector_anis` in the exmaple) function and the `sphere_point` helper function (import sphere-quad.h for it to work). In `eff_chi1_inv`, now replace the line where it calls `chi1p1` with the stretching function times `chi1p1`. Do the same in the `normal_vector_anis` routine, adding an additional argument for the component direction in the function header and changing the function call in `eff_chi1_inv` accordingly. I actually have not thought this through, but this way of calculating a gradient seems weird, multiplying a gradient by a tensor component. It works, however, for now. Additionally, reimplement `has_mu` and let it answer `true` because of the transformation.

Now for the source transformation. In general, to get plane waves, a volume source is necessary. If the volume extends over several resolution domains, the source amplitude in the different domains has to be adjusted. While this is possible in meep by giving `add_volume_source` a function pointer, it is not convenient for the given implementation, i.e. using a class that handles the resolution domains. We want to give `add_volume_source` a member function of our variable-resolution handler class instead of a function pointer, two totally different things in C++. Converting the latter into the former is nearly impossible and, if done, unreadably ugly. So instead, I changed the meep library source files and recompiled. Specifically, I changed the function header of `add_volume_source` so it takes a `std::function<complex<double>(const vec &)>` object instead of a function pointer. This assures backwards compatibilty (the function object can be initialised with a function pointer) and every member function can be cast into this std::function with std::bind, at least if the compiler is not at least a decade outdated and knows its C++11. The compiler reminded me to make the same replacement in the struct `source_vol_chunkloop_data`. Important at this point is that the source amplitudes are, for whatever reason, given in a grid volume centered coordinate system. Even if normally the lower left corner is at $(0, 0, 0)$, now that vector points to the middle of the system, apparently. Correcting for that is, however, easy.

Scattering Spectra

Scattering spectra are usually calculated with `dft_flux_box`-es. This method calculates of the frequency-dependent energy flux through each point of a surface enclosing the scatterer. One calculation without the scatterer is done to get the flux of caused by the incoming wave. Afterwards, another calculation including the scatterer has to done and the flux of the incident wave from the former calculation is substracted at each point. Finally, the energy flux is integrated over the surface, giving the overall scat-

tering intensity, which afterwards has to be normalized with regard to the frequency-dependent exitation intensity, i.e. the Furier-transform of the incoming pulse.

The question in the context of a variable-resolution geometry is, whether the flux through a surface is invariant under the given coordinate transform. We can show that it is indeed, meaning that the routine described above can be applied in the variable-resolution calculations without any modifications.

We want to show the following equality:

$$\iint_{\Omega} \mathbf{S}(\omega) \cdot \mathrm{d}\mathbf{A}_{\Omega} = \iint_{\Omega'} \mathbf{S'}(\omega) \cdot \mathrm{d}\mathbf{A'}_{\Omega} \quad , \tag{1}$$

with the surface Ω, the time-averaged Poynting vector $\mathbf{S}(\omega) = \frac{1}{2}\mathbf{E}(\omega) \times \mathbf{H}^*(\omega)$, the vector area element (normal vector times scalar area element) on the the surface $\mathrm{d}\mathbf{A}_{\Omega}$. Primed quantities are the transformed quantities according to S. Johnson's reference. The surface Ω is given by a parametrization $\varphi(u, v)$. The vector area element is

$$\begin{aligned} \mathrm{d}\mathbf{A}_{\Omega} &= \varphi_u \times \varphi_v \, \mathrm{d}u \, \mathrm{d}v \\ \varphi_u &= \frac{\partial \varphi}{\partial u} \end{aligned} \tag{2}$$

Changing to index notation and employing Einstein's sum convention, we get

$$\begin{aligned} 2 \iint_{\Omega} \mathbf{S}(\omega) \cdot \mathrm{d}\mathbf{A}_{\Omega} &= \iint_{\Omega} \varepsilon_{abc} E_a H_b^* \mathrm{d}A_c \\ &= \iint_{\Omega} \varepsilon_{abc} E_a H_b^* \, \varepsilon_{mnc} \varphi_{m_u} \varphi_{n_v} \, \mathrm{d}u \, \mathrm{d}v \\ &= \iint_{\Omega'} \varepsilon_{abc} \mathcal{J}_{ia} E'_i \mathcal{J}_{jb} H'^*_j \, \varepsilon_{mnc} \mathcal{J}^{-1}_{mx} \varphi'_{x u} \mathcal{J}^{-1}_{ny} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \\ &= \iint_{\Omega'} \varepsilon_{abc} \mathcal{J}_{ia} E'_i \mathcal{J}_{jb} H'^*_j \, \delta_{cd} \, \varepsilon_{mnd} \mathcal{J}^{-1}_{mx} \varphi'_{x u} \mathcal{J}^{-1}_{ny} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \\ &= \iint_{\Omega'} \varepsilon_{abc} \mathcal{J}_{ia} E'_i \mathcal{J}_{jb} H'^*_j \, \delta_{dc} \, \varepsilon_{mnd} \mathcal{J}^{-1}_{mx} \varphi'_{x u} \mathcal{J}^{-1}_{ny} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \\ &= \iint_{\Omega'} \varepsilon_{abc} \mathcal{J}_{ia} E'_i \mathcal{J}_{jb} H'^*_j \, \mathcal{J}^{-1}_{d\alpha} \mathcal{J}_{\alpha c} \, \varepsilon_{mnd} \mathcal{J}^{-1}_{mx} \varphi'_{x u} \mathcal{J}^{-1}_{ny} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \\ &= \iint_{\Omega'} \varepsilon_{abc} \mathcal{J}_{ia} \mathcal{J}_{jb} \mathcal{J}_{\alpha c} E'_i H'^*_j \, \varepsilon_{mnd} \mathcal{J}^{-1}_{mx} \mathcal{J}^{-1}_{ny} \mathcal{J}^{-1}_{d\alpha} \varphi'_{x u} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \\ &= \iint_{\Omega'} \varepsilon_{ij\alpha} \det(\mathcal{J}) E'_i H'^*_j \, \varepsilon_{xy\alpha} \det(\mathcal{J}^{-1}) \varphi'_{x u} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \\ &= \iint_{\Omega'} \varepsilon_{ij\alpha} E'_i H'^*_j \, \varepsilon_{xy\alpha} \varphi'_{x u} \varphi'_{y_v} \, \mathrm{d}u' \, \mathrm{d}v' \quad , \end{aligned} \tag{3}$$

which is just Eq. 1. Here, δ_{cd} denotes the Kronecker-delta. In the process, we used the fact that

$$\varphi_{m_u} = \frac{\partial \varphi_m}{\partial u} = \frac{\partial \varphi_m}{\partial \varphi'_x} \frac{\partial \varphi'_x}{\partial u} = J^{-1}_{mx} \varphi'_{x u}$$

as well as $\varepsilon_{abc} \mathcal{J}_{ia} \mathcal{J}_{jb} \mathcal{J}_{kc} = \varepsilon_{ijk} \det(\mathcal{J})$, which is implied by the definition of the determinant of 3×3 matrices $\varepsilon_{abc} \mathcal{J}_{1a} \mathcal{J}_{2b} \mathcal{J}_{3c} = \det(\mathcal{J})$, and the well known relations $\det(A) = \det(A^T) = 1/\det(A^{-1})$.

Example files in this folder

The example files contain:

- a 2D and a 3D example geometry with a dielectric sphere (cylinder) in the middle, pmls and point source.
- a 2D example with a dielectric cylinder illuminated by a plane wave source. The folder also contains the changed meep source files.
- a 2D example that calculates the scattering of a gold cylinder using `dft_flux_box`.

I did not include any evaluation routines.