other AFM utilities ⬇

A FM data importers for Blender

Blender (link to site) is a ray-tracing program that can be used to render AFM data with shadows and reflections.

---

download the Blender bcrf importer and the Blender gsf importer from https://github.com/AFMiJ/Blender-Utilities

Versions ending with 3 are compatible with Blender version 3 *(tested with Blender 3.1.2)*, versions ending with 2 are compatible with Blender version 2 *(tested with Blender up to 2.79)*.

Installation notes:

- **Blender version 3**: donwload the zip files, then open the preferences viewer: select the *Add-ons* tab (on the left), then click the *Install* button (top). Navigate to the downloaded zip file, select it, finally click on the *Install Add-on* button. Search on the list the new importer and tick the small square button to enable it. Should appear in the *File/Import* menu.
- **Blender version 2**: donwload the zip files, then install the importers through the blender interface: *File/User Preferences* – then select the *Addons* tab, click on the *Install from File...* button. Navigate to the downloaded zip file, select it, finally click on the *Install from File...* button. Tick the small square enable button. Should appear in the *File/Import* menu.

When importing data, a z scale enhancing factor can be adjusted to improve the appearance of the object (default value is 3).

(Note 1: there is basically no control on the imported file type except for the file extension. The file extension must be .bcrf for bcrf file type and .gsf for the gsf file type.)

(Note 2: the core code of these importers is based on the ascii importer developed by zeffii – link bcrx.)

(Note 3: these importers are released under the GPL licence V3.)

B lender color scale shaders

To render the imported AFM meshes with a height depending color scale use the Open Shading Language (OSL) with the following settings in Blender:

1. Rendering engine: use Cycles Render
2. Activate the Open Shading Language in the render settings tab
3. Write a script using the Open Shading Language to define a material with a color that depends on the z coordinate and save it as text file with the .osl extension. Alternatively, use the ImageJ macro *OSL_Generator.ijm* (download from //github.com/AFMiJ/Blender-Utilities, but if you use AFMiJ instead of plain ImageJ it is already included as menu command) to generate an OSL script from a *Gwyddion* color gradient (below is an example of an OSL script generated from an AFM-like yellow colorscale).
4. Use the Shader Editor (with Nodes) to assign the OSL script file to a material (add a script node, then connect the output of the script node to the input of a Material Output node (Blender V3), or a Diffuse BSDF node (Blender V2), see figure below). The script node contains two parameters to adjust the color scale to the object height and position.
5. Assign the material to the AFM object
6. Adjust the parameters (in the node editor) to fit the actual z position and z range of the object with the color scale (the absolute z coordinate is used in the script).

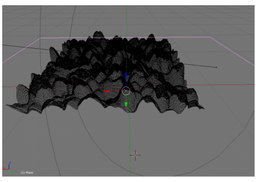
(Note 4: tested with Blender v2 (several versions), and v3, checked with version 3.1.2.)

An OSL script node material for Blender V2. Click on the External tab to select the script. Use a Diffuse BSDF node between the script and the Material Output node	An OSL script material for Blender V3. Click on the External tab to select the script.
	

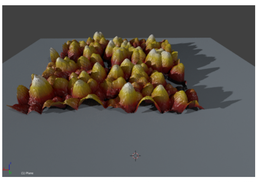
Below an example of a material node obtained mixing a z depending colorscale (from the OSL script) with a glossy white material.

An AFM material in the Node editor. In this example the AFM height dependent color is mixed with a glossy white material.

Below an example of rendering.



a nanostrucured surface imported in Blender and visualized as wireframe



the same surface rendered as an object with an OSL material, lights and shadows

Example of OSL script:

```
//OSL script generated by OSL_Generator version=0.3
shader AFM (
float Zshift = 0.0,
float Z_factor = 3.0,
float rr[6] = {0.2, 0.45, 0.6, 1.0, 0.5, 1},
float gg[6] = {0.2, 0.45, 0.3, 0.9, 0.5, 1},
float bb[6] = {0.2, 0.45, 0.5, 1.0, 0.0, 0},
float hh[6] = {0, 0.3625, 0.564, 4.875, 4.89, 15},
output color Color = 0
)
{ int arlength = arraylength(rr);
float delta = 0;
float dz = 0;
float r = 0;
float g = 0;
float b = 0;
float Z_nm = (P[2] / Z_factor) * 100 - Zshift; /* from internal blender coords to nm.
Inversion of the formula used in the data importer */
if (Z_nm<=hh[1] ) {
r=rr[1];
g=gg[1];
b=bb[1];
}
for (int i = 2;i <= arlength-1; ++i) {
if (Z_nm>hh[i-1] && Z_nm<=hh[i])
{ delta = (hh[i]-hh[i-1]);
dz = Z_nm-hh[i-1];
r = dz * ( rr[i] -rr[i-1] ) / delta + rr[i-1];
g = dz * ( gg[i] -gg[i-1] ) / delta + gg[i-1];
b = dz * ( bb[i] -bb[i-1] ) / delta + bb[i-1];
break;
}
}
if (Z_nm>=hh[arlength])
{ r = rr[arlength];
g = gg[arlength];
b = bb[arlength];
}
color C = color ("rgb", r,g,b);
Color= C;
}
```