User guide

Editor: Viktor Bernholtz

Version 0.2

Status

Granskad	DOK	2018-05-25
Godkänd		

PROJECT IDENTITY

2018/Spring, Group 2
Linköpings Tekniska Högskola, IFM

Group members

Name	Role	Phone no.	E-mail address
Anders Rehult	Project leader (PL)	076-3161206	andre449@student.liu.se
Marian Brännvall	Document manager (DOK)	070-7280044	marbr639@student.liu.se
Andreas Kempe	Secretary (SE)	073-9796689	andke133@student.liu.se
Viktor Bernholtz	Viktor Bernholtz (VB)	073-0386030	vikbe253@student.liu.se

Client: IFM, Linköpings universitet, 581 83 Linköping
Contact person of client: Rickard Armiento, 013-281249, rickard.armiento@liu.se
Course examinator: Per Sandström, 013-282902, persa@ifm.liu.se
Main supervisor: Johan Jönsson, 013-281176, johan.jonsson@liu.se

Contents

Document history

Version	Date	Changes	Done by	Reviewed
0.1	2018-05-22	First draft.	Project group	PL
0.1	2018-05-25	Second draft.	Project group	DOK

1 Introduction

ENVISIoN is an open source toolkit for electron visualisation.

ENVISIoN is implemented by using a modified verision of the Inviwo visualisation framework, developed at the Scientific Visualization Group at Linköpings universitet, LIU.

ENVISIoN has been developed as a part of the course TFYA75: Applied Physics - Bachelor's Project, given at Linköpings universitet, LiU.

The present version was developed during the spring term of 2018 by a project group consisting of: Anders Rehult, Marian Brännvall, Andreas Kempe and Viktor Bernholtz. Supervisor: Johan Jönsson; Requisitioner and co-supervisor: Rickard Armiento; Visualization expert: Rickard Englund; and Course examiner: Per Sandström. The work is based on a previous version by the project group taking the course in the spring term of 2017 consiting of: Josef Adamsson, Robert Cranston, David Hartman, Denise Härnström, Fredrik Segerhammar. Supervisor: Johan Jönsson; Requisitioner and co-supervisor: Rickard Armiento; Visualization expert: Peter Steneteg; and Course examiner: Per Sandström.

ENVISIoN provides a set of Python scripts that allow the user to:

- Read and parse output from electronic structure calculations made by the program VASP and storing the result in a structured HDF5 file.
- Generate interactive Inviwo visualisation networks for common tasks when analyzing electronic structure calculations. Presently there is support for visualising the crystal structure of the unit cell of a material, electron localization function (ELF)-data, electronic charge density, and density of states - both total and partial. The system also provides the ability to interconnect some of the networks mentioned above.

2 How to build and run ENVISIoN

This veriosn of ENVISIoN has been developed on Ubuntu Linux 16.4 and 17.10.

2.1 Creating working directory

```
mkdir -p ~/ENVISIoN
cd ~/ENVISIoN
```

2.2 Install dependencies

Git:

```
sudo apt-get install git
```

Dependencies for Inviwo:

```
sudo apt-get install build-essential

sudo apt-get install cmake cmake-qt-gui cmake-curses-gui

sudo apt-get install libpython3-dev libpython3-dbg

sudo apt-get install mesa-common-dev libglu1-mesa-dev

sudo apt-get install libxcursor-dev libxinerama-dev libxrandr-
    dev
```

Qt5, this version of ENVISIoN has been developed on Qt5.6.1:

```
wget http://download.qt.io/official_releases/qt/5.6/5.6.1/qt-
    opensource-linux-x64-5.6.1.run
chmod +x qt-opensource-linux-x64-5.6.1.run
[ "$XDG_SESSION_TYPE" == "wayland" ] && xhost si:localuser:root
     # enable sudo with gui if on Wayland
sudo ./qt-opensource-linux-x64-5.6.1.run
[ "$XDG_SESSION_TYPE" == "wayland" ] && xhost -si:localuser:
    root
qtchooser -install Qt5.6.1 /opt/Qt5.6.1/5.6/gcc_64/bin/qmake
export QT_SELECT=Qt5.6.1
```

Dependencies for ENVISIoN:

```
sudo apt-get install doxygen python-sphinx-rtd-theme
sudo apt-get install python3-h5py python3-regex python3-numpy
    python3-matplotlib
```

2.3 Download ENVISIoN

```
git clone https://github.com/rartino/ENVISIoN
```

2.4 Download and setup Inviwo

```
git clone https://github.com/inviwo/inviwo.git inviwo.git
```

Prepare Inviwo repository with ENVISIoN patches:

```
cd inviwo.git
git checkout 5fa20ed7d63e9468f437ddefcb06440ffd7db04c
git submodule update --init --recursive
patch -p1 < ../ENVISIoN/inviwo/patches/
    layerrampprecision_swizzleswap.patch
patch -p1 < ../ENVISIoN/inviwo/patches/hdf5_module_elseif.patch
patch -p1 < ../ENVISIoN/inviwo/patches/
    pyvalueparser_matrix_intvectorproperty.patch
patch -p1 < ../ENVISIoN/inviwo/patches/
    hdf5volumesource_dimensions_no_lower_bound.patch
patch -p1 < ../ENVISIoN/inviwo/patches/makePyList_leak.patch
cd ..
```

Setup the Inviwo build directory:

```
mkdir -p inviwo-envision
cd inviwo-envision
```

Create Makefiles with cmake:

```
cmake -G 'Unix Makefiles' -DCMAKE_PREFIX_PATH="/opt/Qt5
    .6.1/5.6/gcc_64/lib/cmake" -DIVW_DOXYGEN_PROJECT=OFF -
    DIVW_MODULE_PYTHON3=ON -DIVW_MODULE_PYTHON3QT=ON -
    DIVW_PROFILING=ON -DIVW_MODULE_BASECL=OFF -
    DIVW_MODULE_OPENCL=OFF -DIVW_MODULE_NIFTI=OFF -
    DIVW_MODULE_VECTORFIELDVISUALIZATION=ON -
    DIVW_MODULE_VECTORFIELDVISUALIZATIONGL=ON -DIVW_CMAKE_DEBUG=
    OFF -DIVW_EXTERNAL_MODULES="$(pwd -P)/../ENVISIoN/inviwo/
    modules" -DIVW_MODULE_CRYSTALVISUALIZATION=ON -
    DIVW_MODULE_GRAPH2D=ON -DIVW_MODULE_HDF5=ON -
    DIVW_MODULE_QTWIDGETS=ON -DCMAKE_CXX_FLAGS="-isystem /opt/
    Qt5.6.1/5.6/gcc_64/include/QtWidgets -isystem /opt/Qt5
    .6.1/5.6/gcc_64/include/" ../inviwo.git
```

Perform the build, set $x = n + 1$ where n is the number of cores:

To check number of cpu cores:

```
cat /proc/cpuinfo
```

```
make -jx
```

3 Start Inviwo and run ENVISIoN scripts

From the Inviwo build directory:

```
bin/inviwo
```

- Open Python Editor under Python menu.
- In the Python editor, click Open Script.
- Select one of the scripts.
- Click open.
- Click the python logo in the top left corner to run.

More information about each script and how to use them in chapter 5.

4 How to develop ENVISIoN and Inviwo

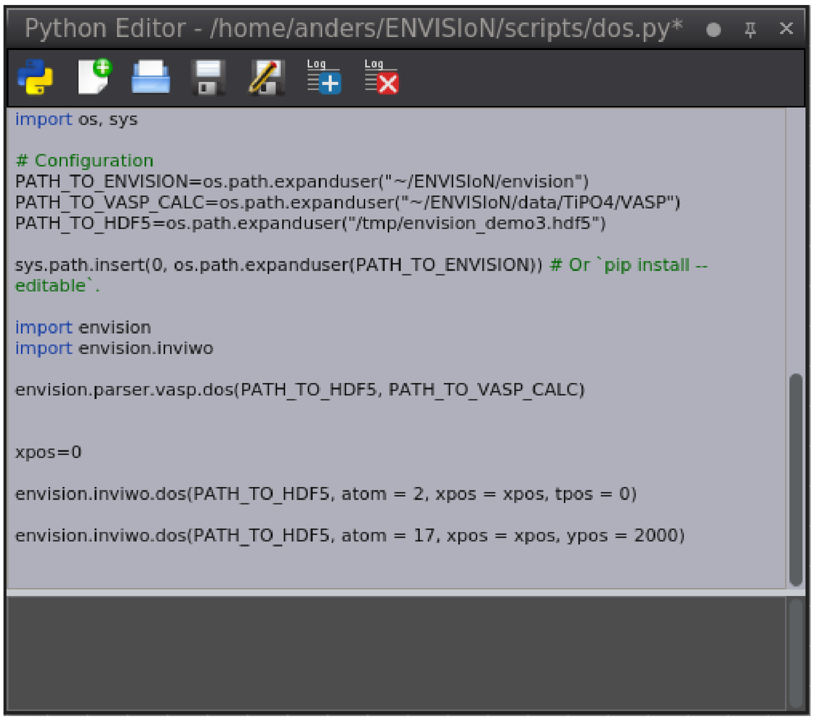
A development environment is already installed, namely Qt5.

Create a Qt5 cmake projekt:

```
mkdir −p
```

5 Scripts

Descriptions and examples of all the avaliable scripts are found below. The scripts are called through the python editor in Inviwo as shown in figure 1.


Python Editor - /home/anders/ENVISIoN/scripts/dos.py*

```
import os, sys

# Configuration
PATH_TO_ENVISION=os.path.expanduser("~/ENVISIoN/envision")
PATH_TO_VASP_CALC=os.path.expanduser("~/ENVISIoN/data/TiPO4/VASP")
PATH_TO_HDF5=os.path.expanduser("/tmp/envision_demo3.hdf5")

sys.path.insert(0, os.path.expanduser(PATH_TO_ENVISION)) # Or `pip install --
editable`.

import envision
import envision.inviwo

envision.parser.vasp.dos(PATH_TO_HDF5, PATH_TO_VASP_CALC)


xpos=0

envision.inviwo.dos(PATH_TO_HDF5, atom = 2, xpos = xpos, tpos = 0)

envision.inviwo.dos(PATH_TO_HDF5, atom = 17, xpos = xpos, ypos = 2000)
```

Figure 1: Python script for calling on the DOS visualisation module of ENVISIoN. Note that the pathway PATH_TO_VASP_CALC here is set to "~/ENVISIoN/data/TiPO4/VASP", the folder containing the relevant VASP output files.

In all of the scripts the user has to specify the path to ENVISIoN, a path where VASP output files are found, and a desired path where ENVISIoN will generate a HDF5-file, or, if one already exists, use that file and skip parsing VASP data. **NOTE:** change the value of PATH_TO_HDF5, delete the HDF5-file, or rename the HDF5-file in between visualisations if a new material is to be visualised.

5.1 Unit cell

In the Python editor of Inviwo the user can choose to generate a visualisation of the unit cell by opening the folder *scripts* and running the file *unitcell.py*.

Figure 2 is an example of a unit cell visualisation, in this case of TiPO4. Figure 3 shows the network responsible for the aforementioned visualisation.

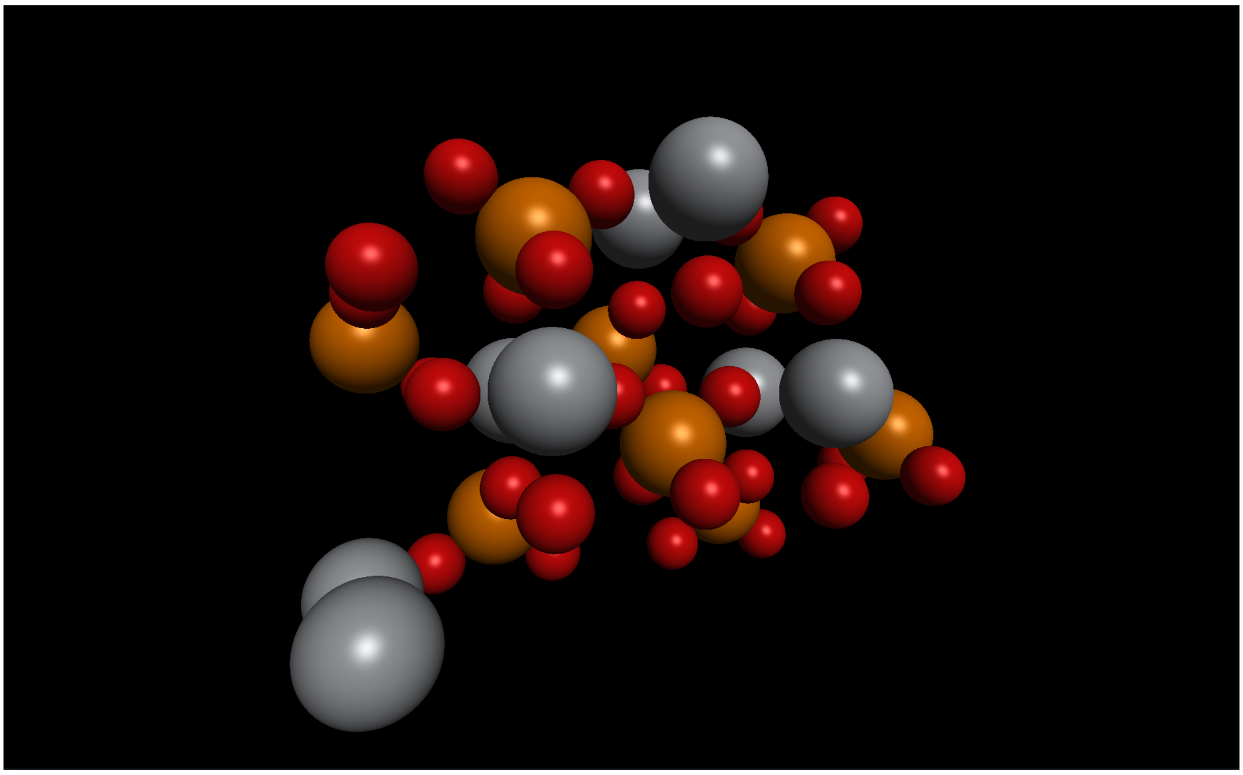


Figure 2: Visualisation of the crystal structure of the unit cell of TiPO4 in Inviwo.

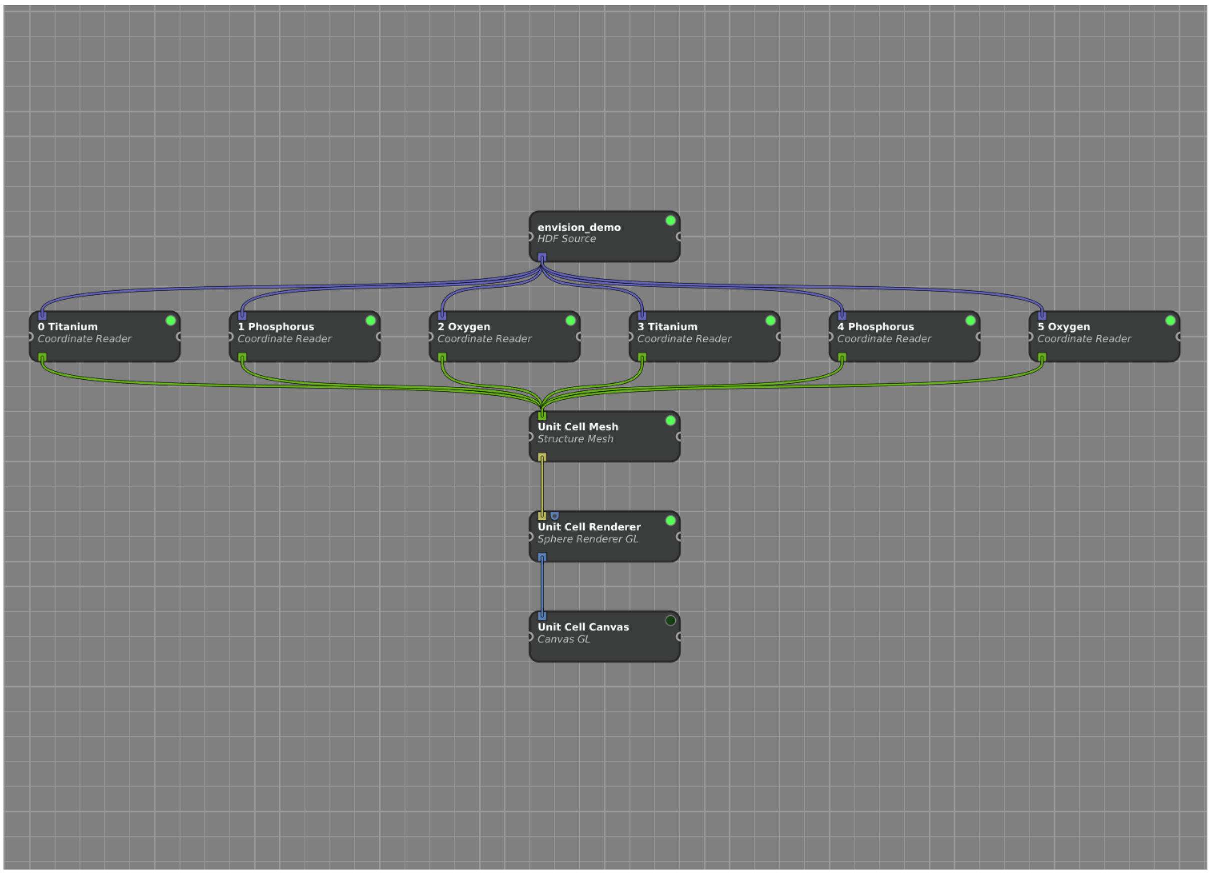


Figure 3: Network for visualisation of TiPO4 in Inviwo.

5.2 ELF

In the Python editor of Inviwo the user can choose to generate a visualisation of the Electron Localisation Function (ELF)-data as shown in figure 4 with corresponding network shown in figure 5 by opening the folder *scripts* and running the file *elf.py*. A second image showing the ELF-data of a slice of the unit cell can be added by changing the argument *Slice* of the function *elf* to True. This slice corresponds to a plane intersecting the unit cell, which can be moved using the W and S keys on the keyboard. The orientation of this plane can be changed by selecting the processor "Volume Slice" in the network by clicking on it, then choosing another value from the drop down menu in the property "Slice along axis".

The ELF-data can also be visualised as an isosurface. This is achieved by changing the argument *iso* of the function *elf* from None to any value between 0 and 1. The slice function described above is not compatible with isosurface visualisation.

The colors and opacities of different values can be changed by selecting the processor "ELF raycaster" in the network, clicking on the property "Transfer function", and interacting with the window that pops up. This window shows multicolored dots which can be manipulated to control opacity. Dots can be added by double clicking, and removed by clicking a dot and pressing the Delete key on the keyboard. Dots can be dragged and dropped using the mouse. The horizontal axis of the plane on which the dots are placed corresponds to ELF values between 0 and 1, while the vertical axis corresponds to opacities between 0 (transparent) and 1 (completely opaque). Figure 9 in chapter 5.3 shows what this window looks like.

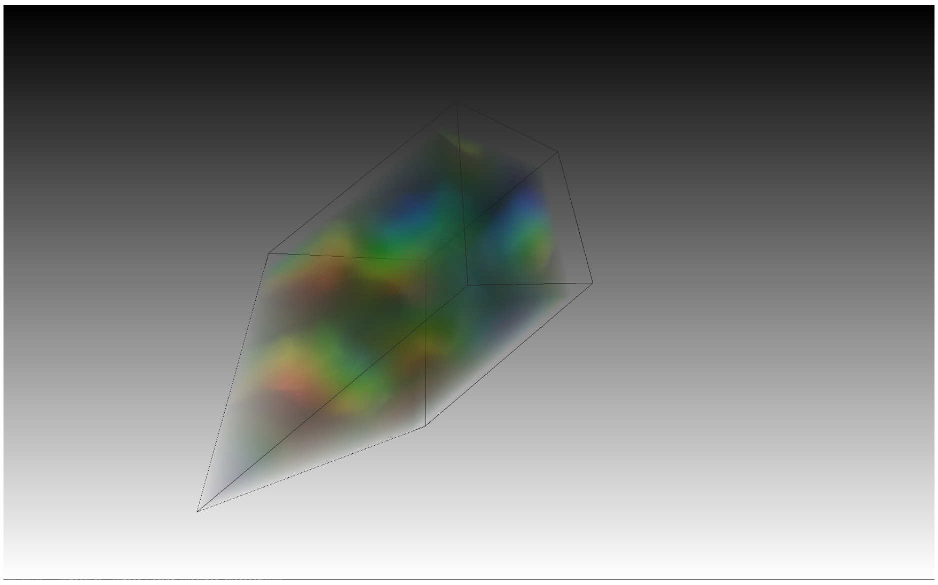


Figure 4: Visualisation of ELF-data for diamond in Inviwo.

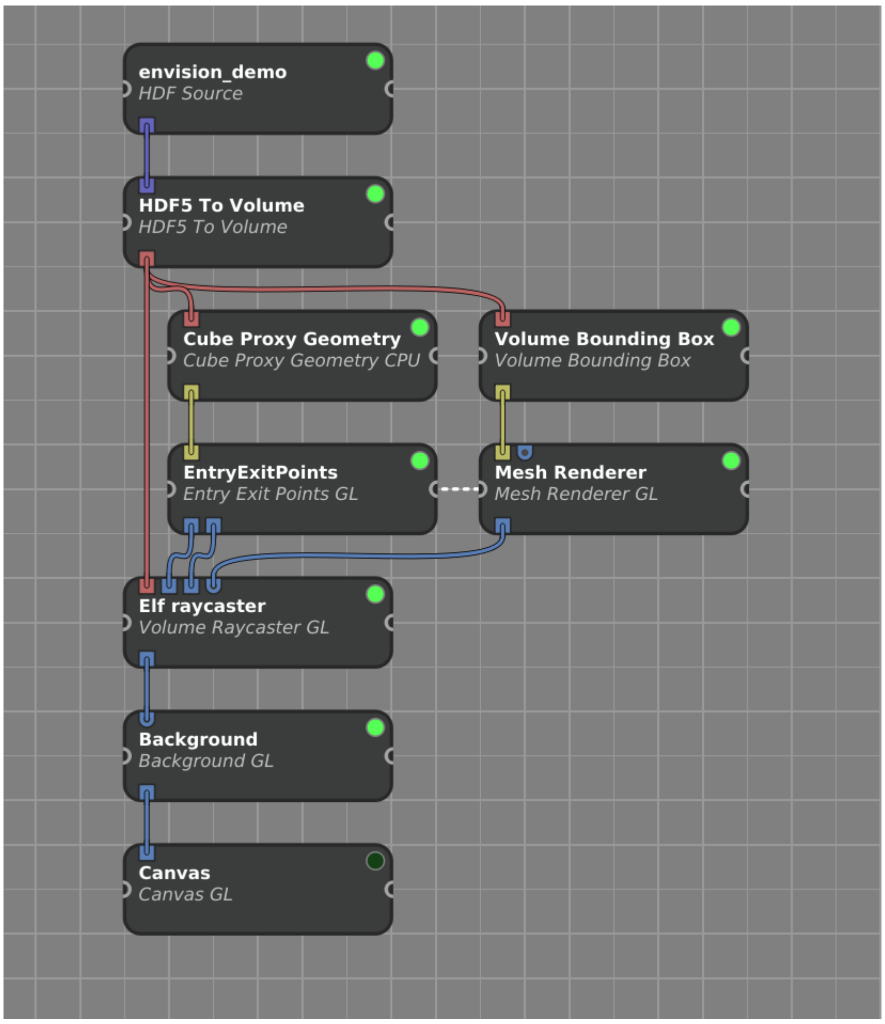


Figure 5: Network for visualisation of ELF-data for diamond in Inviwo.

5.3 Charge density

In the Python editor of Inviwo the user can choose to generate a visualisation of the charge density by opening the folder *scripts* and running the file *charge.py*. A second image showing the charge density of a slice of the unit cell can be added by changing the argument *Slice* of the function *charge* to True. This slice corresponds to a plane intersecting the unit cell, which can be moved using the W and S keys on the keyboard. The orientation of this plane can be changed by selecting the processor "Volume Slice" in the network by clicking on it, then choosing another value from the drop down menu in the property "Slice along axis". This type of visualisation is shown in figure 6, and the generating network is found in figure 7.

The charge density can also be visualised as an isosurface. This is achieved by changing the argument *iso* of the function *charge* from None to any value between 0 and 1. Figure 8 shows an example of this. The slice function described above is not compatible with isosurface visualisation.

The colors and opacities of different values can be changed by selecting the processor "Charge raycaster" in the network, clicking on the property "Transfer function", and interacting with the window that pops up. This window shows multicolored dots which can be manipulated to control opacity. Dots can be added by double clicking, and removed by clicking a dot and pressing the Delete key on the keyboard. Dots can be dragged and dropped using the mouse. The horizontal axis of the plane on which the dots are placed corresponds to charge density values between 0 and 1, while the vertical axis corresponds to opacities between 0 (transparent) and 1 (completely opaque). Figure 9 shows what this window looks like.

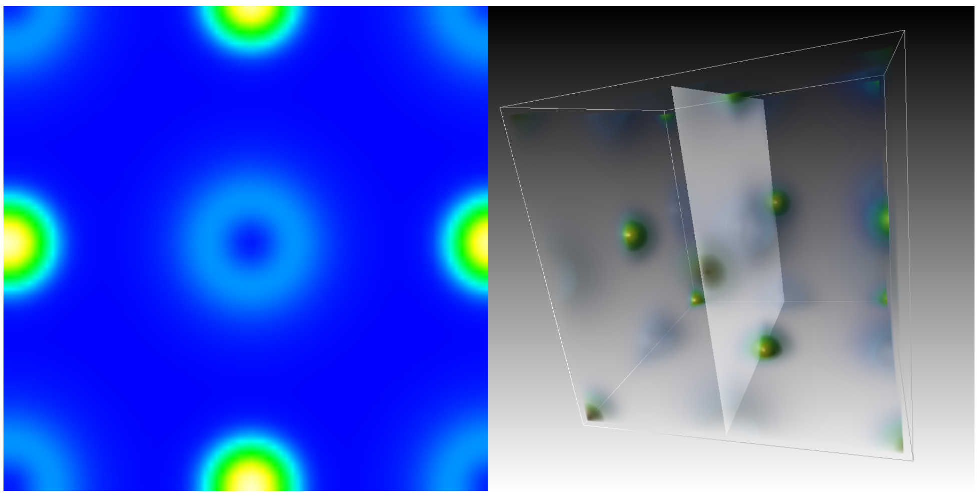


Figure 6: Visualisation of charge density for NaCl with slice function in Inviwo.

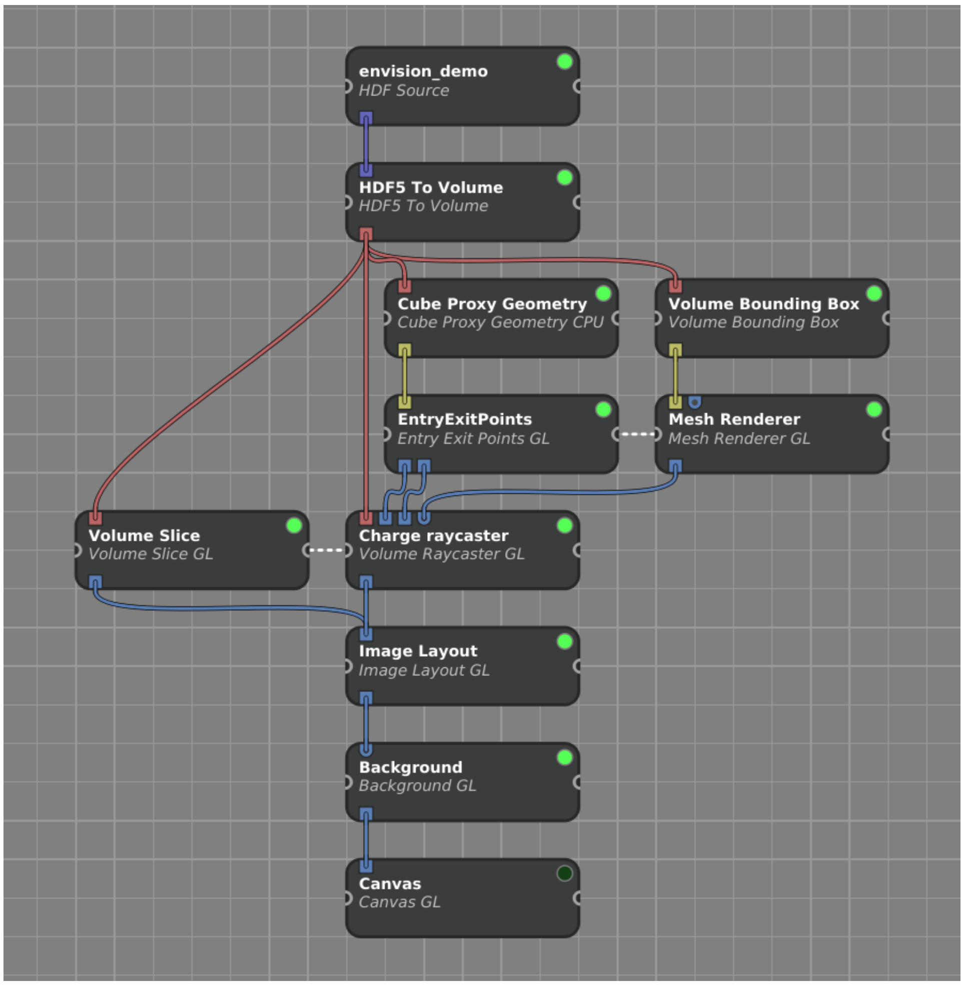


Figure 7: Network for visualisation of charge density for NaCl with slice function in Inviwo.

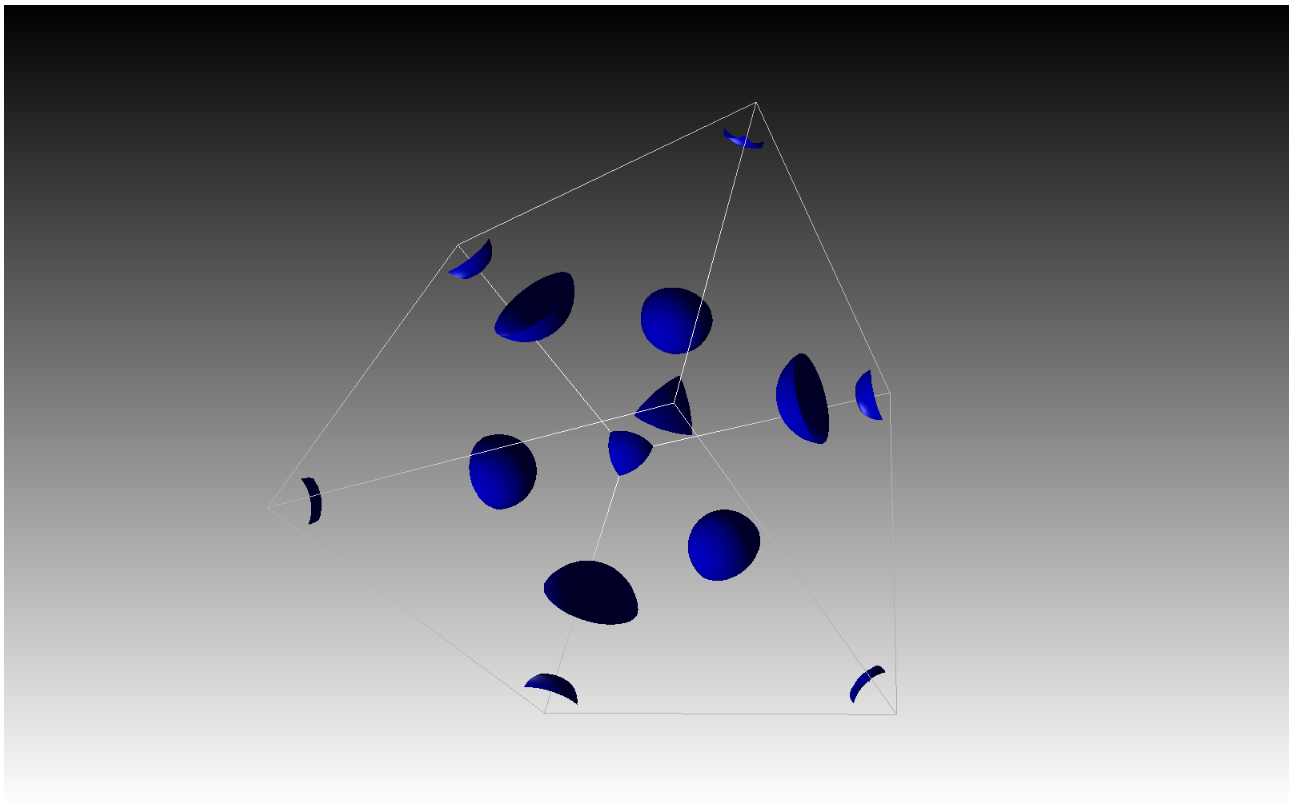


Figure 8: Visualisation of charge density for NaCl with ISO raycasting.

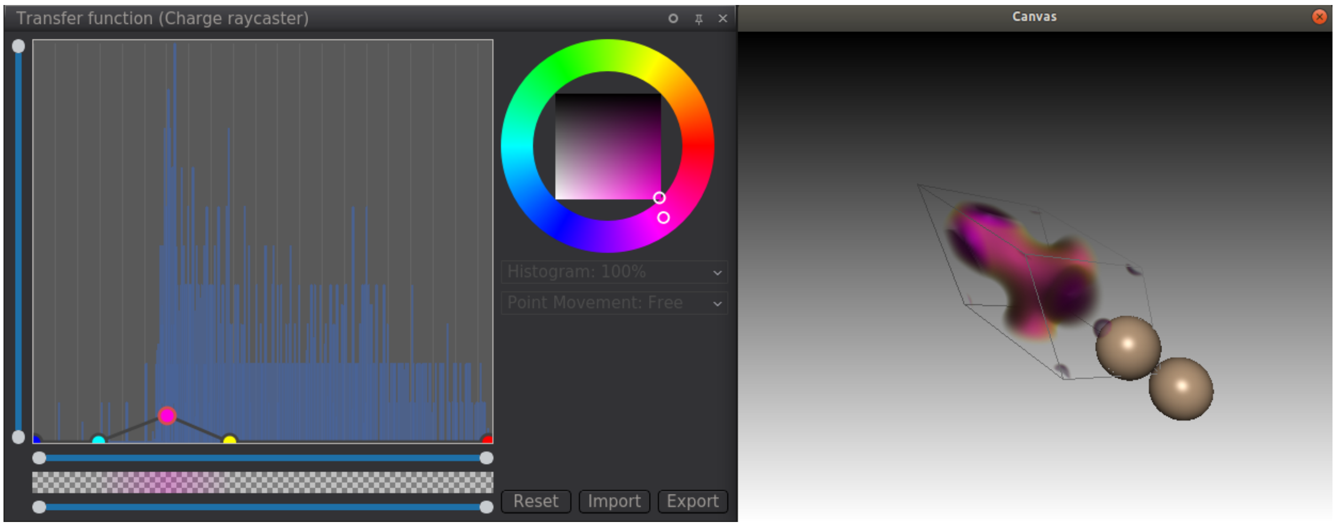


Figure 9: Transfer function of the Charge raycaster processor and visualisation of charge density of Si connected with unit cell visualisation. Charge density values between 0.15 and 0.45 are illustrated as a somewhat transparent purple volume.

5.4 DOS

In the Python editor in Inviwo, the user can choose to generate a visualisation of the density of states by opening the folder *scripts* and running the file *dos.py*.

Figure 10 is an example of a visualisation of the total density of states, in this case for Cu. Figure 11 shows the network for the visualisation. This network generates both a 2D-graph of the total density of states and a similar graph but for the partial density of states. The partial density of states is by default shown for one atom, in two graphs for spin-polarized calculations, determined by the argument *atoms* in the function *dos*. See chapter 5.5.3 for a description of how to pick the atom for which to show projected DOS.

Visualisation of the partial DOS for one specific orbital (s, p, d, or f) of an atom can be achieved by deleting the connections between the undesired processors of type "HDF5 To Function" and the corresponding processor with name "Partial Add". To do this, select a connection by clicking on it and press the Delete key.

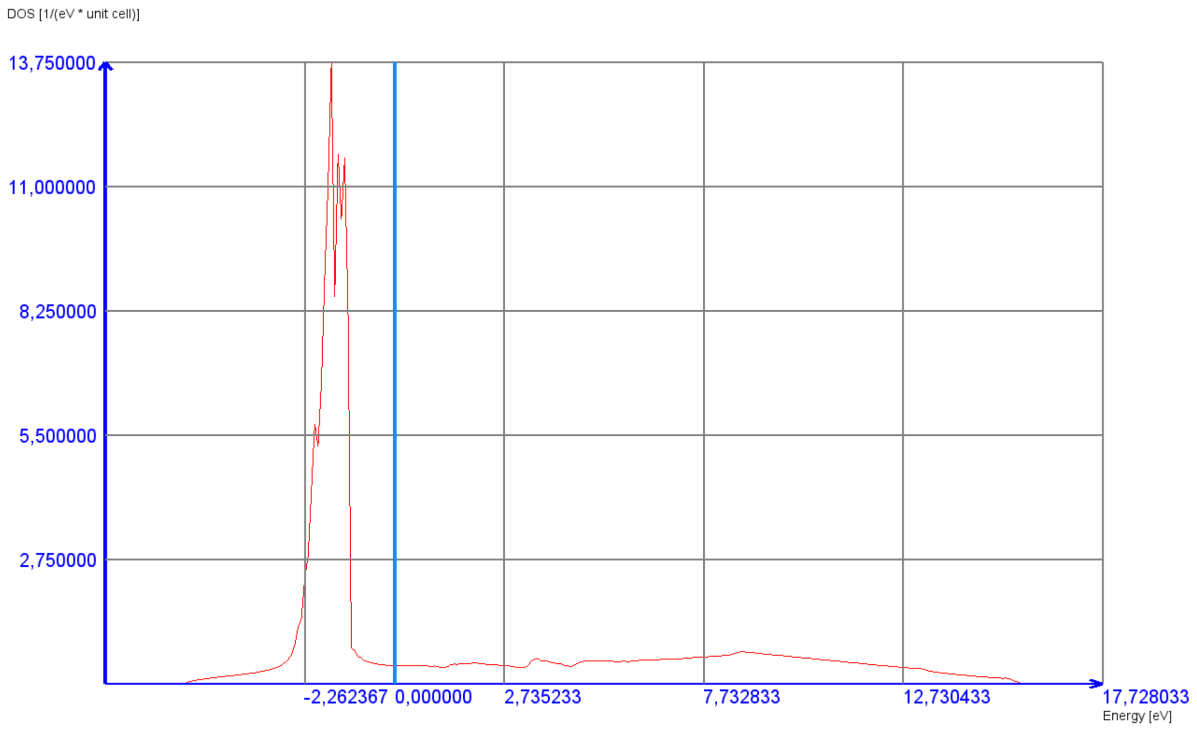


Figure 10: Visualisation of total DOS for Cu.

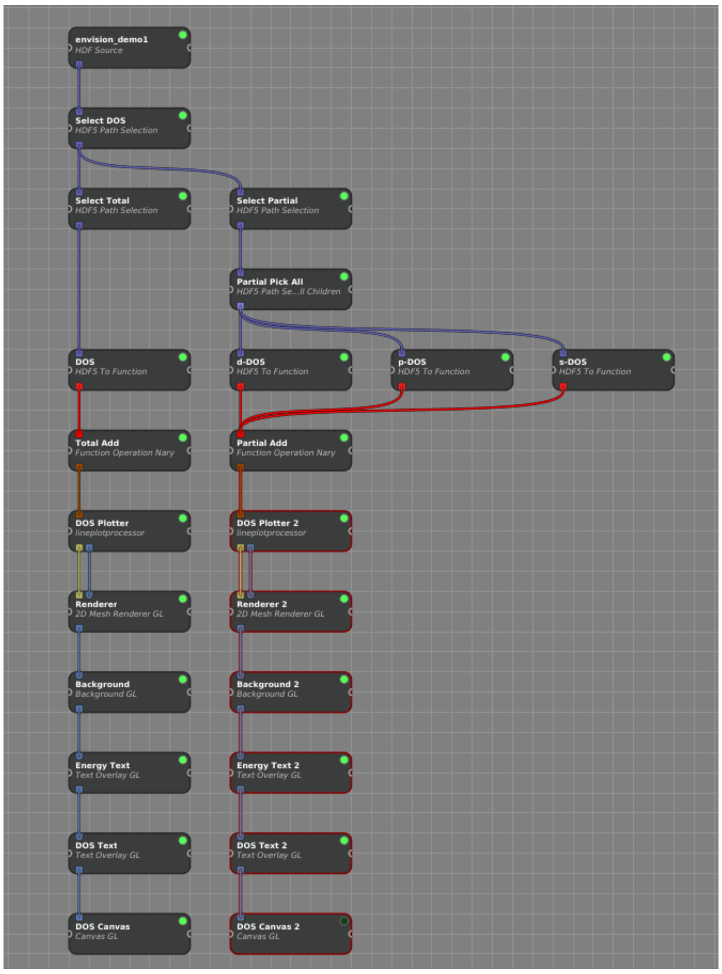


Figure 11: Network for visualisation of total and partial DOS for Cu.

5.5 Interconnected networks

Descriptions and examples of the python scripts generating interconnected networks can be seen below.

5.5.1 Unit cell and charge

In the Python editor in Inviwo, the user can choose to generate a visualisation of unitcell and charge by opening the folder *scripts* and running the file *unitcellAndCharge.py*.

5.5.2 Unit cell and ELF

In the Python editor in Inviwo, the user can choose to generate a visualisation of unitcell and ELF by opening the folder *scripts* and running the file *unitcellAndElf.py*. Figure 12 shows what this can look like.

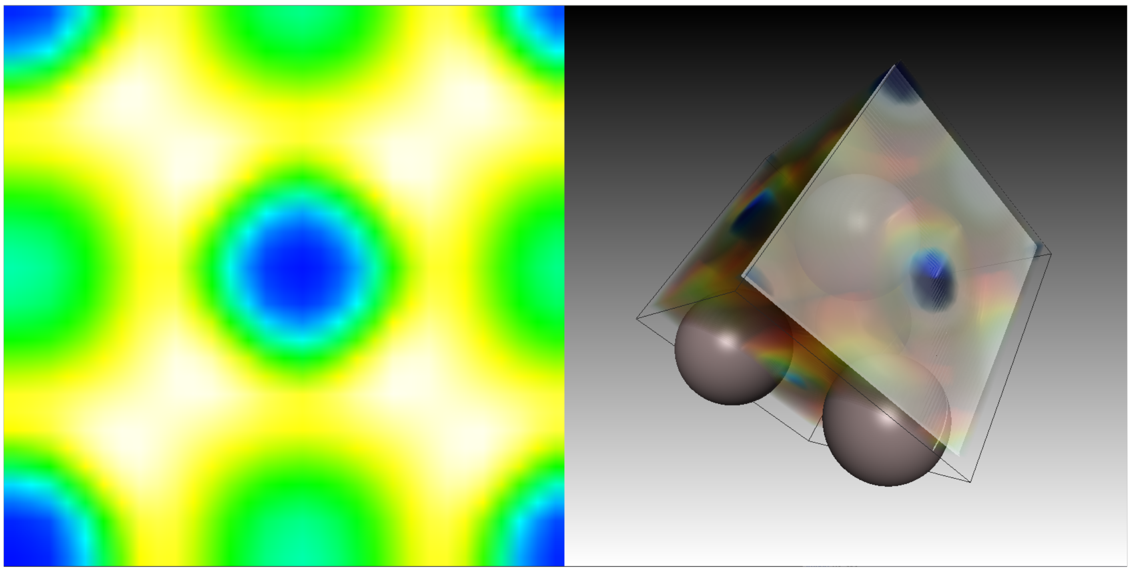


Figure 12: Unit cell and ELF data for aluminum visualised side by side, with volume raycasting and slice function.

5.5.3 Unit cell and DOS

In the Python editor in Inviwo, the user can choose to generate a visualisation of unitcell and density of states by opening the folder *scripts* and running the file *unitcellAndDos.py*. Here the user can pick an atom for which to show partial DOS by clicking on the desired atom in the unit cell. See figure 13 for an example of what this can look like.

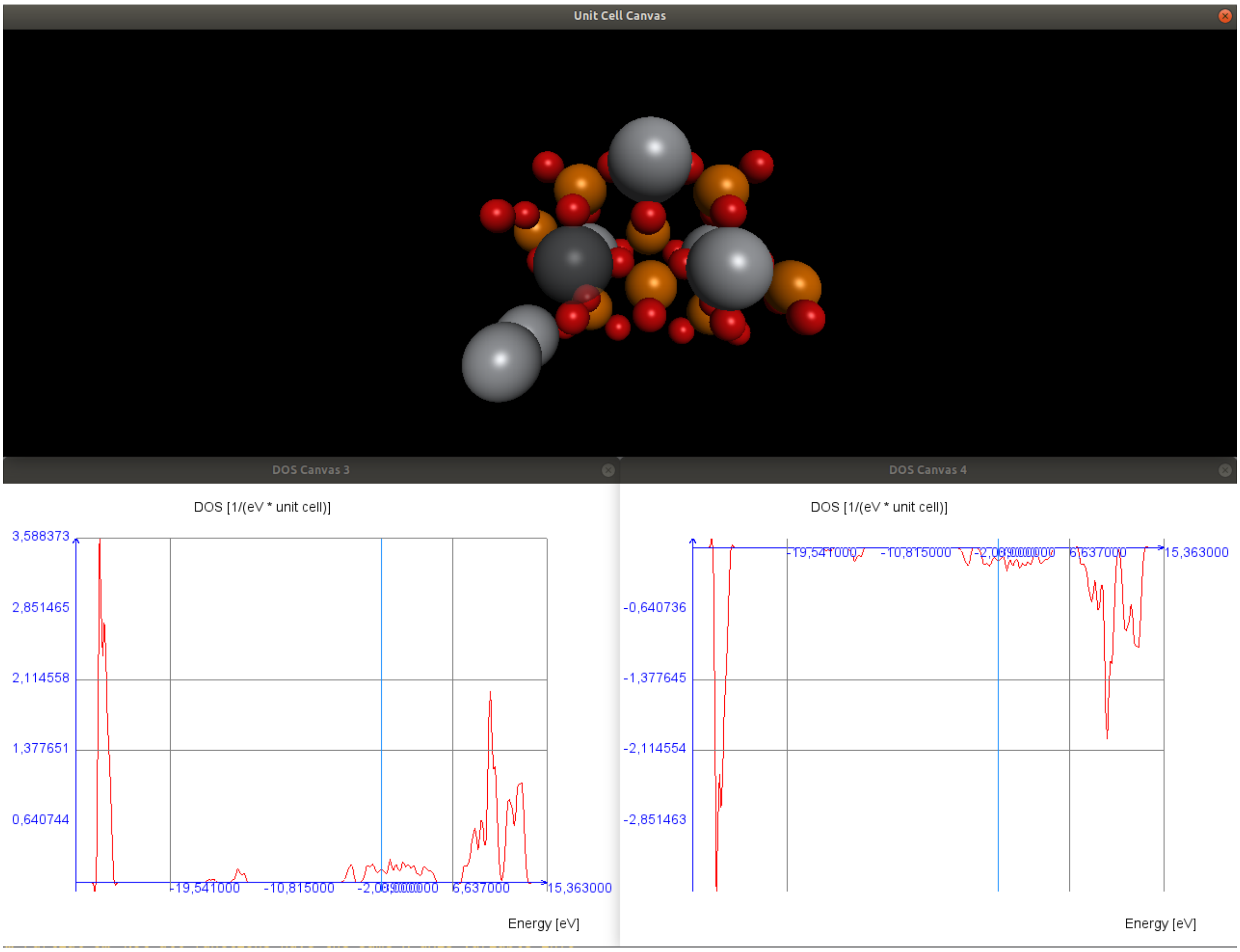


Figure 13: Visualisation of unit cell data and partial DOS of a titanium atom. The two graphs show DOS for electron spin up and electron spin down.