

Analysing the Molecular Geometry of Disordered Structures using Mogul and the CSD Python API (MOG-003)

Developed using
2025.2 CSD Release

Table of Contents

Introduction.....	2
Learning Outcomes	2
Pre-required Skills	2
Materials.....	2
Example 1. Analysing the Geometry of a Disordered Structure in Mercury	3
Part 1. Selecting Disorder Assemblies and Groups	3
Part 2. Running Mogul Geometry Check.....	5
Conclusion	7
Example 2. Analysing the Geometry of a Disordered Structure using the CSD Python API	8
Part 1: Exploring a Disordered Structure in the CSD Python API	8
Part 2: Analysis of Molecular Geometry	10
Extension: Iterating over Disorder Combinations	15
Conclusion	16
Summary.....	20
Next Steps.....	20
Feedback.....	20
Glossary	21
Basics of Mercury Visualization	22

Introduction

Mogul is the CSD software for analysing molecular geometry; it is available as a stand-alone program and is additionally exposed through the Mercury interface and via the CSD Python API. It can compare bond lengths and angles, torsions and ring geometry in a molecule with distributions calculated from hundreds of thousands of measurements in the CSD. One relevant application of Mogul is in assessing the performance of a disorder model from a structure refinement. 3D disorder information is coded into the CSD for over 300,000 entries and individual disorder configurations can be accessed in Mercury and the CSD Python API in over half of these.¹ In this workshop you will see how to access and process a disordered structure with Mogul Geometry Check in Mercury and with the CSD Python API.

Learning Outcomes

After completing this workshop, you will be able to:

- Identify and select disorder assemblies and groups in Mercury
- Run a Mogul Geometry Check on a disordered structure in Mercury
- Select disorder assemblies and groups in the CSD Python API and run Mogul Geometry Check

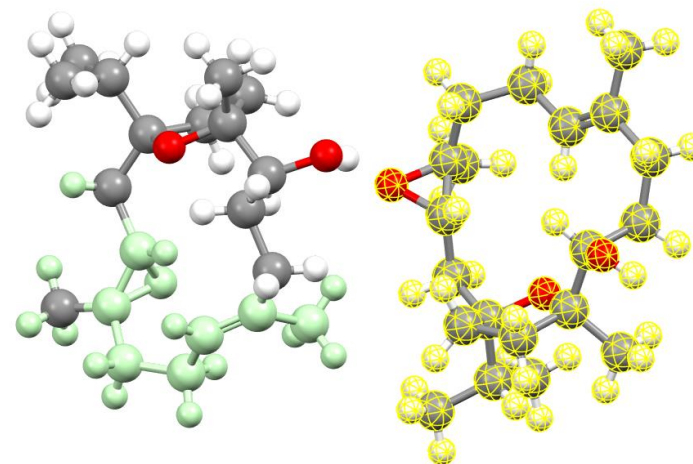
This workshop will take approximately **1.5 hours** to be completed. The words in *Blue Italic* in the text are reported in the [Glossary](#) at the end of this handout.

Pre-required Skills

For Exercise 1, some familiarity with Mercury is desirable but not essential. For Exercise 2, familiarity with Python is essential and familiarity with the basics of the CSD Python API is desirable. We recommend attempting workshop PYAPI-001 available from the CSD-Core workshops webpage here:

<https://www.ccdc.cam.ac.uk/community/training-and-learning/workshop-materials/csd-core-workshops/>

¹ As of CSD version 6.0. + 1 update

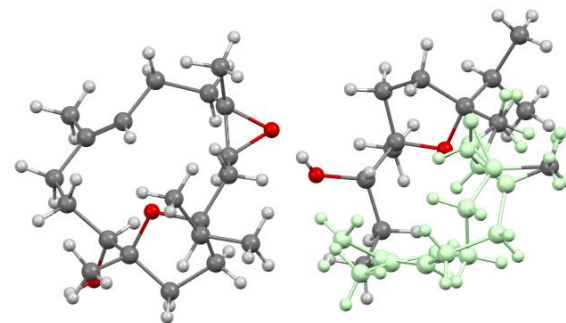


Materials

The Python script for Example 2 can be found [here](#).

Example 1. Analysing the Geometry of a Disordered Structure in Mercury

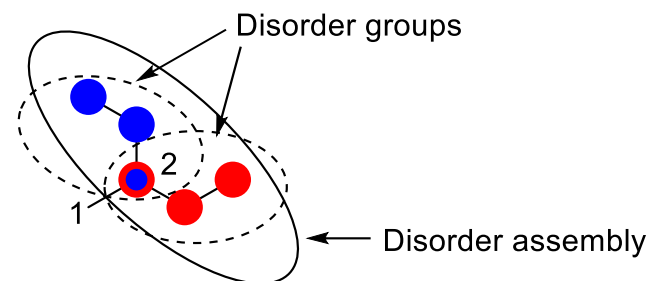
In this example we will first explain the way in which disorder is represented in 3D structures in the CSD and see how to select different assemblies and groups using a pair of stereoisomers of incensole oxide, a natural product found in frankincense resin. We will then see how to run a Mogul Geometry Check on the disordered structure.



CSD entry PURZIO showing one disordered molecule with disordered atoms in green.

Part 1. Selecting Disorder Assemblies and Groups

We begin with some definitions. A disordered structure lacks long range order but may show local order, which may be modelled by the crystallographer during a structure refinement. Many structures in the CSD include the disorder model from the [deposited data](#). A disorder assembly is a cluster of atoms that show long-range positional disorder but are locally ordered. Within each assembly a disorder group is used to identify sites that are simultaneously occupied (i.e. the atoms in the group all have the same [occupancy](#)). A scheme of this is shown for the simple case of a disordered butyl group, with disorder starting at the 2-position, to the right (2 is included because the hydrogens on the 2-carbon which aren't shown would be at different positions in the two groups). We shall now explore examples in the CSD.

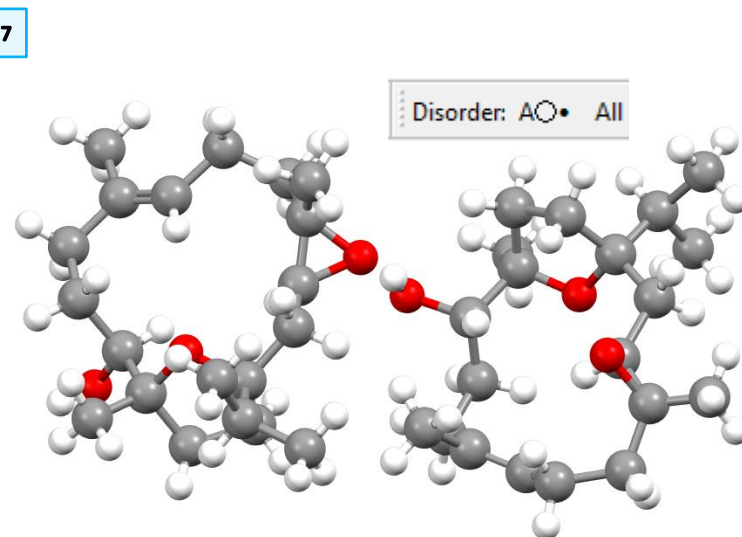
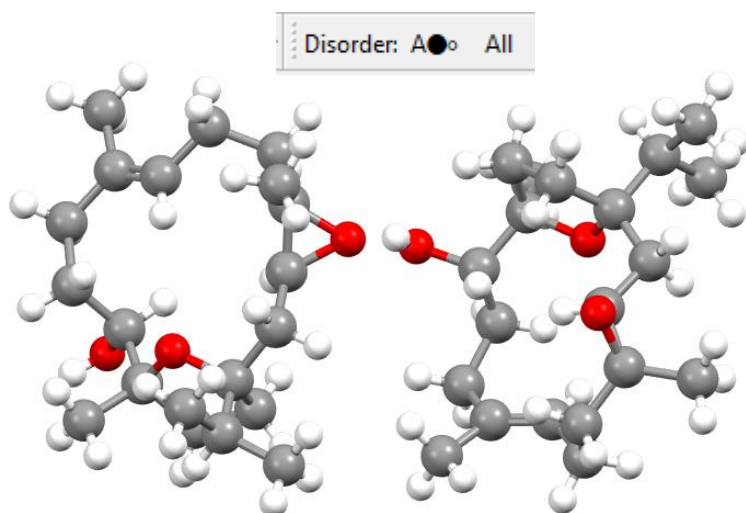
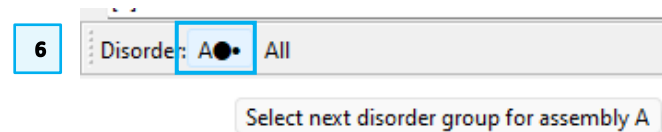
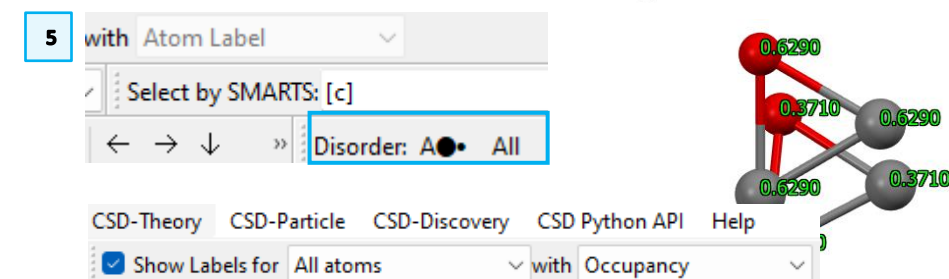
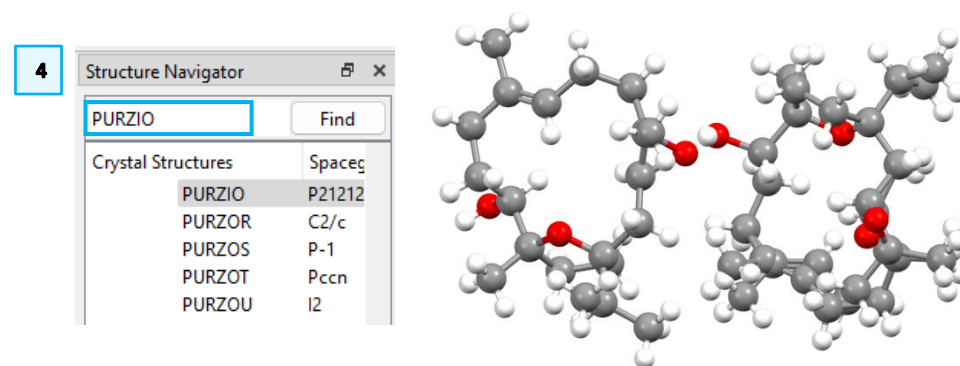


1. Open Mercury from the Start menu or by clicking on the desktop icon.
2. In the *Structure Navigator*, type "INCNSL01". This is one stereoisomer of incensole oxide.
3. This structure is not disordered and in the top toolbar you should see Disorder: None.

Structure Navigator

Crystal Structures	Spaceg
INCNSL01	P21212
INDAEP	Pbnm
INDANT	P21/a
INDAZL	P21
INDAZL01	Cmc21
INDAZL02	P21
INDRIJA	P-1

- In the *Structure Navigator* type "PURZIO". This is a stereoisomer of INCNSL01. It contains two molecules in the [asymmetric unit](#), one of which is disordered.
- The structure contains one disorder assembly consisting of two groups. This is indicated in the top toolbar in the *Disorder* section. The letter *A* is the label for the disorder assembly and the two dots that follow it represent the two disorder groups. The unequal size of these dots indicates that the occupancies are unequal; the larger dot represents the higher occupancy group. Both dots are filled in black which indicates that all assemblies/groups are shown. If you like, you can label the atoms with their occupancy by ticking *Show labels for All atoms with Occupancy* to see the ratio is 0.629: 0.371 for the disordered atoms (untick when you have finished).
- Click on the dots to select a disorder group; by default, the first disorder group will be selected. Where the occupancies are unequal, the major occupancy group comes first – this is the case in PURZIO. You will notice that the minor occupancy group disappears from the view. **Tip:** if there are multiple assemblies in a structure, clicking the button for any assembly will automatically select the major occupancy group for all of the remaining assemblies (you can toggle the groups of each assembly separately).
- Click on the disorder assembly **A** again to toggle between the two groups in the assembly. Clicking **All** will reactivate both disorder groups.



Part 2. Running Mogul Geometry Check

Mogul Geometry Check will evaluate the geometry of molecules in the visualiser, or selected molecules or molecular fragments, as desired by the user. However, it is important to have the desired disorder assemblies/groups selected before starting the run. We shall investigate the molecules individually first.

1. Ensure that the major occupancy disorder group for PURZIO is selected in visualiser.
2. To help distinguish the disordered and non-disordered molecules, you can select *Colour: by Element or Disorder*. The disordered portion of one molecule will then appear highlighted pale green.
3. First, select the non-disordered molecule in the visualiser by holding down the shift key and left-clicking on the molecule. It will become highlighted in yellow.
4. From the CSD-Core menu at the top, select **Mogul Geometry Check**.
5. In the *Mogul Search Setting* dialogue, tick Apply filters and tick **Exclude Organometallics** and **Exclude Powder Structures**. Leave all other settings as their default values.
6. Click **Search**.

The image illustrates the steps for running Mogul Geometry Check:

- 1**: A 3D molecular model of PURZIO is shown. A blue arrow points to a specific disorder assembly.
- 2**: A dropdown menu for 'Colour' is shown, with 'by Element or Disorder' selected.
- 3**: The same 3D model is shown, but the disordered portion of one molecule is highlighted in pale green.
- 4**: A screenshot of the CSD-Core menu at the top of the software, with 'Mogul Geometry Check...' highlighted.
- 5**: A screenshot of the 'Mogul Search Settings' dialog box. The 'Apply filters' checkbox is checked. Under 'Search Filter Options', the 'Exclude' checkbox is checked, and the 'Exclude' dropdown is set to 'Organometallics' and 'Powder structures'.

7. When the search has completed you will see a table of results. Note that this molecule is labelled as PURZIO_2 in the section headings. This will come in useful later. Barring the large 13-membered ring, for which there are understandably few fragments to compare with, all other bonds, angles, etc. are classified as **not unusual (with enough hits)**.
8. You can explore the histograms for the measurements by double clicking on the line of interest. The torsions involving the alkene e.g. C7-C8-C9-C10 might make for interesting points of comparison given the alkene's strikingly different orientation in the disordered molecule, which we will examine next. In this molecule, PURZIO_2, the torsion in a well-populated region of the histogram.
9. Close the Mogul histogram window and the *Mogul Results Viewer*.
10. Check that the major disorder assembly is activated in the disorder selector and shift + left-click on the disordered molecule.
11. Run Mogul Geometry check as explained in steps 4–6 and examine the results (the relevant torsion for comparison is C27-C28-C29-C30, see next page). You will see that the equivalent torsion in molecule PURZIO_1 is unusual.
12. Select the minor disorder group using the disorder selector and shift + left-click to select the molecule.
13. Run Mogul Geometry check as in steps 4–6 and examine the results (the relevant torsion for comparison is C45-C46-C29-C30, see next page). You will see that the torsion here is similarly unusual.

Tip: provided you have a disorder group selected you can run Mogul on both molecules simultaneously (just skip the molecule selection step).

7

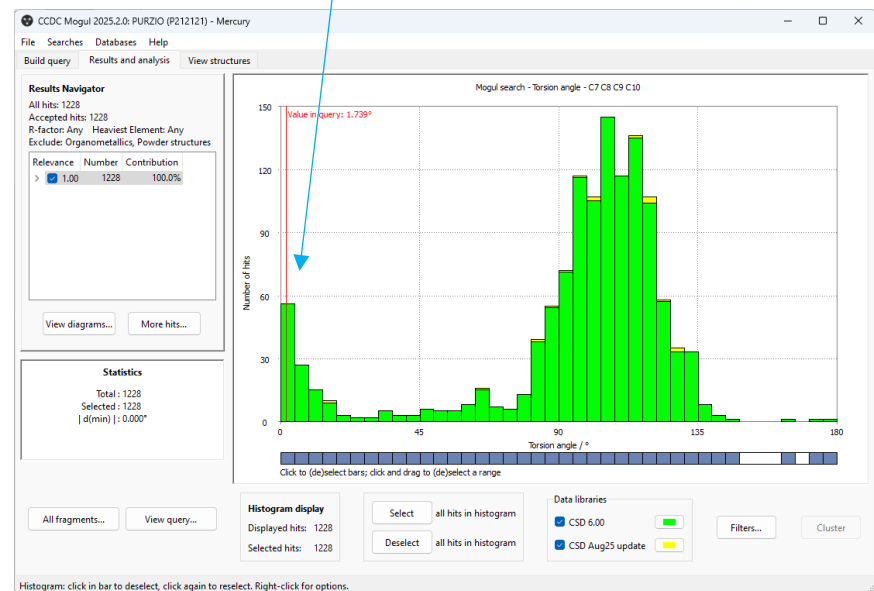
Mogul Results Viewer

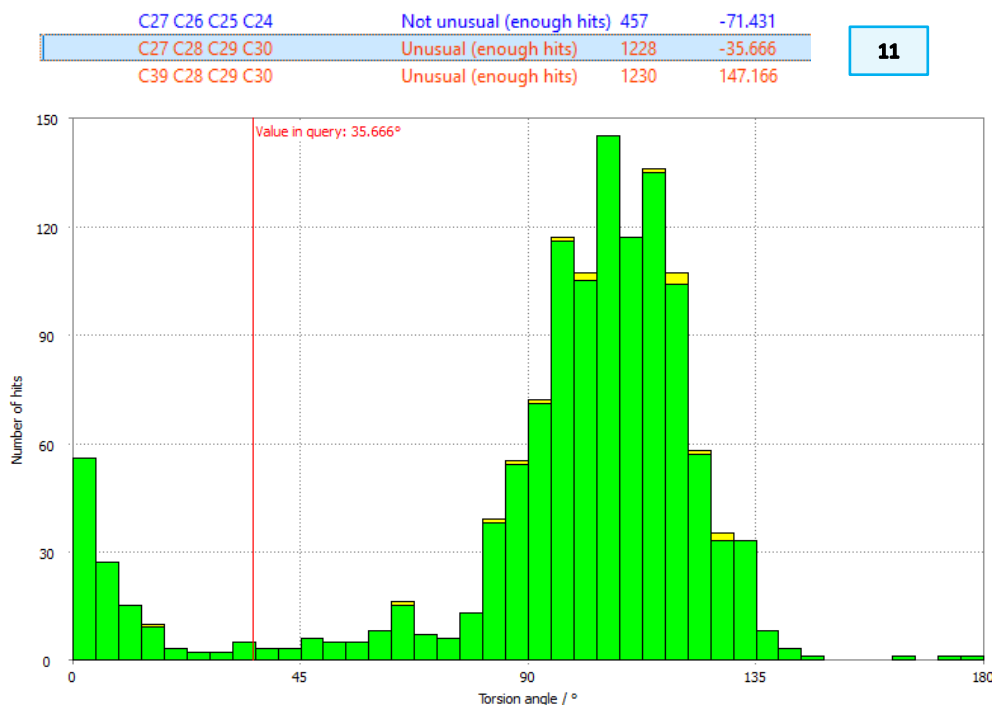
Show / hide: Columns Fragments... Deselect all fragments Export...

Help Double click to view result in Mogul

Type	Molecule	Fragment	Classification	No. of hits	Query value	Mean	Std.
		C16 C15 C1 C2	Not unusual (enough hits)	342	-59.869		
		C16 C15 C1 C14	Not unusual (enough hits)	342	68.714		
		O1 C1 C15 C17	Not unusual (enough hits)	178	-52.226		
		C17 C15 C1 C2	Not unusual (enough hits)	342	65.473		
		C17 C15 C1 C14	Not unusual (enough hits)	342	-165.944		
		C7 C6 C5 C4	Not unusual (enough hits)	452	-63.440		
		C6 C7 C8 C9	Not unusual (enough hits)	924	-178.920		
		C6 C7 C8 C19	Not unusual (enough hits)	967	-0.575		
		C8 C7 C6 C5	Not unusual (enough hits)	1476	-130.569		
		O1 C1 C2 C3	Not unusual (enough hits)	54	-58.189		
		C15 C1 C2 C3	Not unusual (enough hits)	42	-177.458		
		C14 C1 C2 C3	Not unusual (enough hits)	48	55.916		
		O2 C11 C12 O1	Not unusual (enough hits)	90	166.687		
		O2 C11 C12 C13	Not unusual (enough hits)	90	52.895		
		O2 C11 C12 C20	Not unusual (enough hits)	90	-74.540		
		O1 C12 C11 C10	Not unusual (enough hits)	81	-67.140		
		C10 C11 C12 C13	Not unusual (enough hits)	81	179.067		
		C10 C11 C12 C20	Not unusual (enough hits)	81	51.633		
		C7 C8 C9 C10	Not unusual (enough hits)	1213	-1.739		
		C19 C8 C9 C10	Not unusual (enough hits)	1215	179.757		

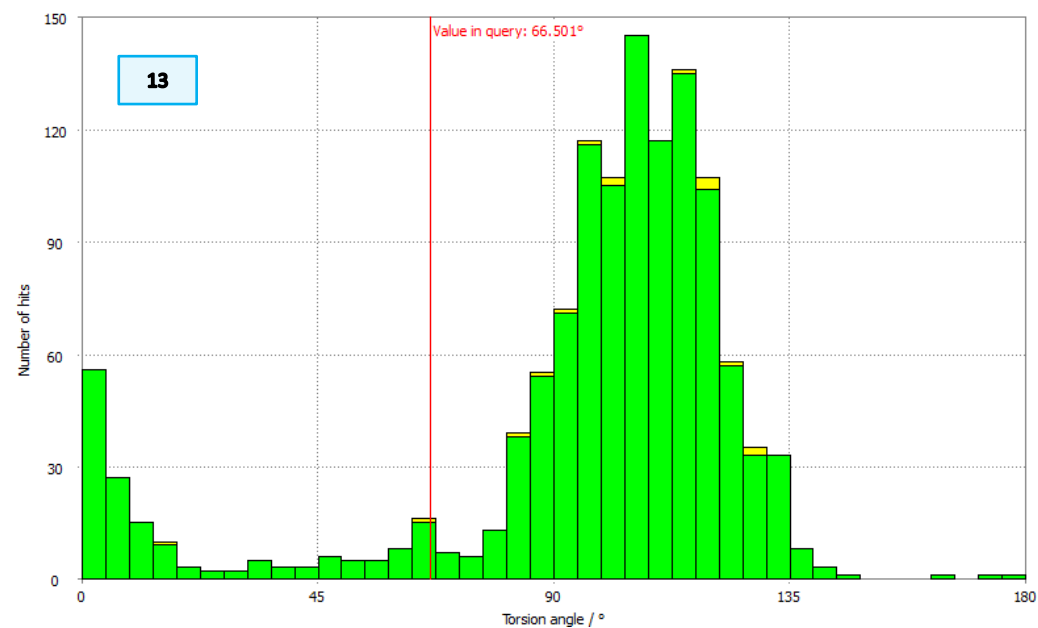
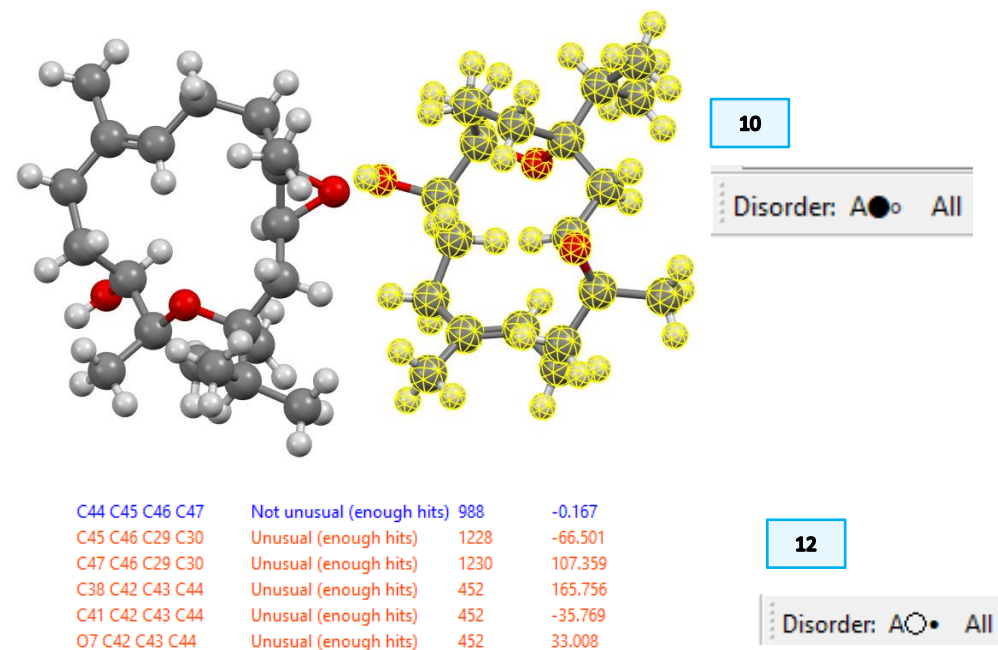
8





Conclusion

In this example we have seen how to select disorder groups in a molecule with one disorder assembly. We have subsequently analysed individual molecules and determined, Using Mogul Geometry Check that the disordered molecule displays unusual torsion values compared to the well-ordered one.

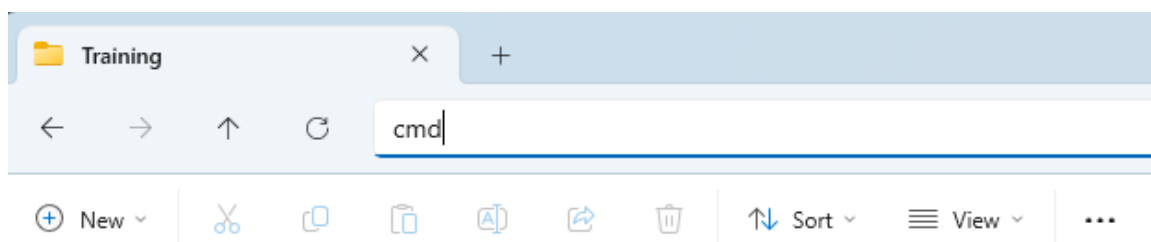


Example 2. Analysing the Geometry of a Disordered Structure using the CSD Python API

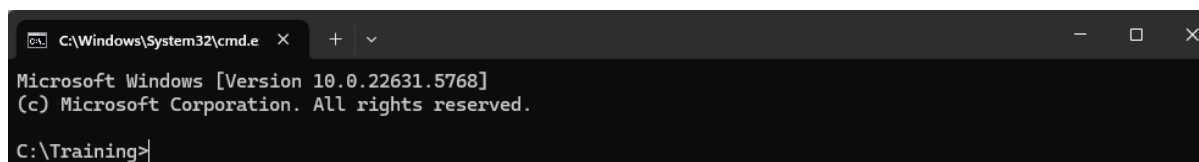
In this example we will begin by exploring the way that disorder is handled in the CSD Python [API](#) using the structure from Example 1, PURZIO. We will then use the geometry analyser in the API to carry out the equivalent function of Mogul. We will write a script that will print the output to the console.

Part 1: Exploring a Disordered Structure in the CSD Python API

1. For this exercise we will be writing the script in a Python file that we can then run from a command prompt later. Start by creating a folder where you will save your Python files in a place where you have read and write access, for example C:\training\ for Windows, or something equivalent on macOS or Linux. We will continue to use our C:\training\ folder (or equivalent), throughout the tutorial.
2. Open the command prompt from this folder. In Windows you can type 'cmd' in the File Explorer tab and press 'Enter'. In Linux you can right click on the folder and select Open in Terminal. In macOS, right click on the folder, select Services then click New Terminal at Folder.

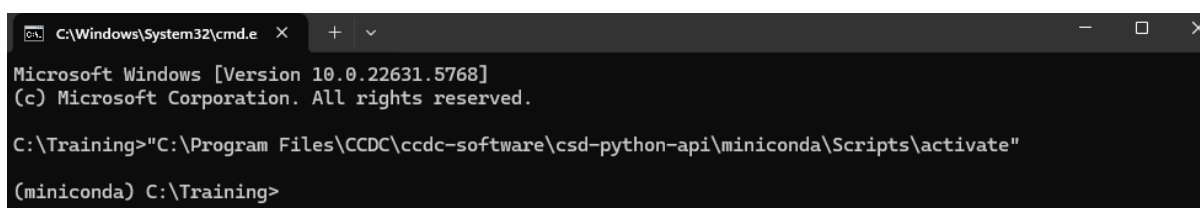


The command prompt should now appear.



3. To run your Python scripts from the command prompt, you will first need to activate your environment. The activation method will vary depending on the platform:
 - **Windows:** Open a command prompt window and type (including the " marks) replacing "path\to" with the exact path to your ccdc-software install.:
`"C:\path\to\CCDC\ccdc-software\csd-python-api\miniconda\Scripts\activate"`
 - **MacOS/Linux:** Open a terminal window and change directory to the CSD Python API bin folder:
`cd /Applications/CCDC/ccdc-software/csd-python-api/miniconda/bin`
 Then activate the environment with:
`source activate`

If the activation is successful, (miniconda) will appear at the beginning of your command prompt:



4. We can now start writing our script. In the folder you created, open your preferred text editor, and create a new Python file called *analyse_disorder.py*. The following steps show the code that you should write in your Python file, along with explanations of what the code does.
5. The CSD Python API makes use of different modules to do different things. The `ccdc.io` module is used to read and write entries, molecules, and crystals. The `ccdc.conformer` module contains classes concerned with molecular conformations. To make use of modules, we first need to import them.

```
from ccdc.io import EntryReader
from ccdc.conformer import GeometryAnalyser
```

6. We begin by reading the PURZIO CSD entry using an instance of the `EntryReader` class. The refcode for a CSD entry can be accessed by the `identifier` property, which we will print for clarity.

```
entry_reader = EntryReader('CSD')
purzio_entry = entry_reader.entry('PURZIO')
print(f'Refcode: {purzio_entry.identifier}')
```

Save the changes to your file and run the Python script in the command line by typing:

```
python analyse_disorder.py
```

You should see the output “Refcode: PURZIO” printed in the console.

7. Next, we need to access the crystal object from the entry. The hierarchical structure of the CSD Python API means that the crystal can be accessed from the entry, the molecule can be accessed from the crystal, and the atoms can be accessed from the molecule.

```
purzio_crystal = purzio_entry.crystal
```

9. We are now at the point where we can examine the disorder. In the structure we have selected, we already know from Exercise 1 that there is disorder, but if we did not already know this, we could find out using the `has_disorder` property.

```
print(f'Crystal has disorder: {purzio_crystal.has_disorder}')
```

Run the script in the command prompt. You should find that the new output is “Crystal has disorder: True”.

10. Now to explore the disorder we first need to access the assemblies. The `assemblies` property returns a tuple of `ccdc.crystal.Crystal.Disorder.Assembly` objects. We can see how many assemblies there are with the following code. Please note that it should all be on one line in your script.

```
print(f'Number of disorder assemblies:
{len(purzio_crystal.disorder.assemblies)}')
```

line 1 of 1

Save the changes and run the script in the console. You should see the additional output “Number of disorder assemblies: 1”.

11. We can now check the disorder groups associated with the disorder assembly with the `groups` property. To your script, add the following:

```
for i, assembly in enumerate(purzio_crystal.disorder.assemblies,
                             start=1):
    print(f'Assembly {i} has {len(assembly.groups)} disorder groups.')
```

line 1 of 2
line 2 of 2

Save the changes and run the script in the console. You should see the new output “Assembly 1 has 2 disorder groups.”

12. To work with a disorder assembly/group, we first need to select it, then it must be activated. Add the following to your script:

```
first_assembly = purzio_crystal.disorder.assemblies[0]
first_group = first_assembly.groups[0]
first_group.activate()
```

line 1 of 3
line 2 of 3
line 3 of 3

The above code selects the first assembly (index 0) and the first disorder group (index 0). The last line uses the `activate()` method to set the specified group as active. To check that this has worked as expected, add the following lines to your script.

```
print(f'Activated assembly_id: {first_assembly.id}')
```

```
print(f'Activated group_id: {first_assembly.active.id}')
```

line 1 of 2
line 2 of 2

Save the changed and run the script. You should get the additional output “Activated assembly_id: 0” and “Activated group_id: 0”. You might also wish to know the occupancy of the disordered atoms in the selected disorder group. This can be ascertained with the following:

```
print(f'Occupancy of disorder group: {first_group.occupancy:.3f}')
```

line 1 of 1

This will return “Occupancy of disorder group: 0.629”. We are looking at the major occupancy group.

Part 2: Analysis of Molecular Geometry

13. Now that we have activated a disorder group, we can begin to work with the molecules in the structure. The way that the molecule object is treated in the CSD Python API means that it can be made up of several independent molecules – whether chemically identical or not – and so separated molecule are classed as *components*. Let’s check how many components there are with the following code.

```
mol = purzio_crystal.molecule
```

```
print(f'Number of molecules in crystal: {len(mol.components)}')
```

line 1 of 2
line 2 of 2

Save the changes and run the script. You should get the new output “Number of molecules in crystal: 2”.

14. You can select a specific component by its index, for example, to access the first component, we would use `molecule.component[0]`. We shall do this but with slightly different code to produce more obvious output, which will be useful for the analysis later. To your script, add the lines:

```

comp_index = 0
comp = mol.components[comp_index]
print(f'Analysing molecule {comp_index + 1} of {len(mol.components)}')

```

line 1 of 3
line 2 of 3
line 3 of 3

Save the changes and run the script. You should see the additional output “Analysing molecule 1 of 2”.

15. Before analysing the geometry of the component molecules, it will be helpful to establish whether they are individually disordered. Remember that a disorder assembly and group apply to the crystal object; if there are two component molecules (as in our case) but only one is disordered. The two disorder groups will both contain one ordered molecule and one disordered molecule, so we need some way of identifying the disordered components. A simple test would be to see if the occupancy of any constituent atoms of a component molecule has occupancy less than one. This will be the case if any part of the molecule is disordered. To your script add the following:

```

if any([atom.occupancy < 1 for atom in comp.atoms]):
    print('This molecule is disordered.')
else:
    print('This molecule is not disordered.')

```

line 1 of 4
line 2 of 4
line 3 of 4
line 4 of 4

Save the changes and run the script. You should now get the additional output “This molecule is disordered.”

If you wanted to identify the disordered atoms explicitly, you could add the following code:

```

print([(atom.label, atom.occupancy) for atom in comp.atoms if
atom.occupancy < 1])

```

line 1 of 1

This will output a list of tuples “[(‘H22A’, 0.629), (‘H22B’, 0.629)...”. Notice that the occupancies of the disordered atoms are the same as the value from `first_disorder_group.occupancy` accessed at the crystal level, as expected.

16. The GeometryAnalyser module in the CSD Python API is the equivalent of Mogul in the desktop software, used in Example 1. First, we need to create an instance of the GeometryAnalyser.

```
engine = GeometryAnalyser()
```

17. As in Mogul, it is possible to modify search settings and apply filters. To make the results comparable with those from Example 1, we will exclude organometallics and powder structures.

```

engine.settings.organometallic_filter = 'organics_only'
engine.settings.powder_filter = True

```

line 1 of 2
line 2 of 2

You can find all the available setting in the [documentation](#).

18. To analyse the current component molecule, add the following code to your script:

```
geom_analysed_mol = engine.analyse_molecule(comp)
```

line 1 of 1

Saving and running the script at this stage will not produce any output on the console. We need to access the measurements from the analysed molecule. As an example, we will identify an unusual torsion. Add the following code to your script:

```
for torsion in geom_analysed_mol.analysed_torsions:           line 1 of 3
    if torsion.unusual and torsion.enough_hits:                line 2 of 3
        print(f'Unusual torsion: {",".join(torsion.atom_labels)}')
Torsion: {torsion.value:.3f} °')                               line 3 of 3
```

In the above code, the atom labels and torsion value (accessed via the `value` property) will only be printed if they are unusual and if there are enough hits. For comparison, this is equivalent to printing only the lines in red from the Mogul results tables that you will have seen in Example 1.

The output resulting from this code should be:

```
Unusual torsion: C27,C28,C29,C30 Torsion: -35.666 °
Unusual torsion: C39,C28,C29,C30 Torsion: 147.166 °
```

You can access the bonds, angles and rings via `analysed_bonds`, `analysed_angles` and `analysed_rings`. The values that you can retrieve for each type of measurement are different (see the Mogul tables in Example 1 and the CSD Python API documentation).

19. The underlying histogram data can be accessed using the `histogram()` method. If you like, you can specify the `bin_size` (default = 40), minimum and maximum (default 0° and 180°, respectively for torsions). We will not give any arguments and use the defaults.

First, we will get a list of the unusual torsions and select one – C39 C28 C29 C30. To your script, add the following lines of code.

```
unusual_tors = [t for t in geom_analysed_mol.analysed_torsions if
t.unusual and t.enough_hits]                                   line 1 of 2

tor_query = next(t for t in unusual_tors if t.atom_labels == ['C39',
'C28', 'C29', 'C30'])                                         line 2 of 2
```

Then, to get the histogram data:

```
counts = tor_query.histogram()                                  line 1 of 1
```

We can now plot the data. You can use your preferred graphing Python package or stick with `matplotlib`, which is distributed together with the CSD Python API. At the beginning of your script, add:

```
import matplotlib.pyplot as plt                                line 1 of 1
```

The below code plots the data and adds a line at the query value (147.166°). Add it to your script, save the changes and run it.

```
n_bins = len(counts)                                           line 1 of 4
bin_min, bin_max = 0, 180                                       line 2 of 4
bin_width = (bin_max - bin_min) / n_bins                        line 3 of 4
bin_edges = [bin_min + i * bin_width for i in range(n_bins + 1)] line 4 of 4
```

Code continued from previous page.

```
plt.figure(figsize=(8, 5))
plt.bar(
    bin_edges[:-1],
    counts,
    width=bin_width,
    edgecolor="black",
    align="edge",
    color="limegreen"
)

# Red line for torsion value
torsion_val = tor_query.value
plt.axvline(
    torsion_val,
    color="red",
    linestyle="--",
    linewidth=2
)

# Add text label above the histogram, slightly shifted
y_max = max(counts)
plt.text(
    torsion_val + 2,      # shift slightly right
    y_max * 0.9,         # place near the top
    f"{torsion_val:.2f}°",
    color="red",
    fontsize=10,
    rotation=90,
    va="bottom"
)

plt.xlabel("Torsion Angle (degrees)")
plt.ylabel("Count of Observation Hits")
plt.title(f"Histogram for torsion {tor_query.atom_labels}")
plt.xlim(bin_min, bin_max)

plt.show()
```

line 1 of 9
line 2 of 9
line 3 of 9
line 4 of 9
line 5 of 9
line 6 of 9
line 7 of 9
line 8 of 9
line 9 of 9

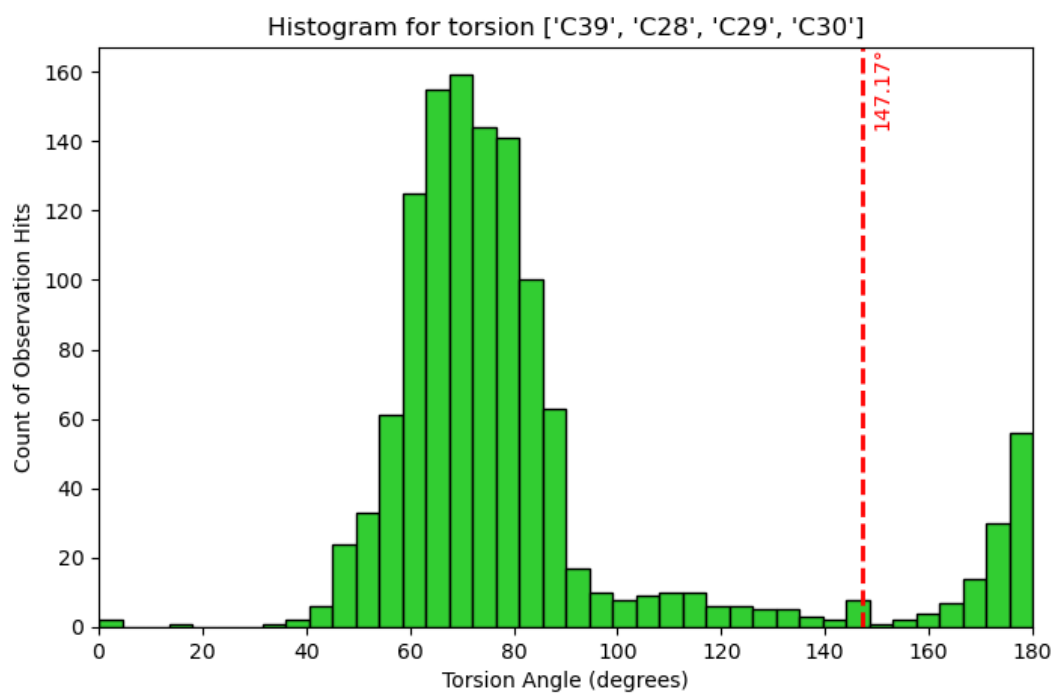
line 1 of 7
line 2 of 7
line 3 of 7
line 4 of 7
line 5 of 7
line 6 of 7
line 7 of 7

line 1 of 10
line 2 of 10
line 3 of 10
line 4 of 10
line 5 of 10
line 6 of 10
line 7 of 10
line 8 of 10
line 9 of 10
line 10 of 10

line 1 of 4
line 2 of 4
line 3 of 4
line 4 of 4

line 1 of 1

The output should look as shown overleaf.



Extension: Iterating over Disorder Combinations

The above example shows you the essential aspects of disorder handling, however, you will typically want to process all possible disorder assembly/group combinations, especially in more complex examples where there might be many more disorder groups/assemblies. The `ccdc.crystal.Crystal.Disorder.combinations` attribute returns an iterator that yields all combinations of the disorder groups. The crystal is updated accordingly with the yielded disorder groups, which provides a convenient way to calculate crystal property taking into account the disorder occupancies.

1. As an example, of how to loop over the disorder combinations to look at the torsion measurements, add the following code to your script.

```
print('\n-----\n')           line 1 of 14
print('Now using disorder combinations\n') line 2 of 14

for comb_id, combination in
enumerate(purzio_crystal.disorder.combinations): line 3 of 14
    for comp_id, component in
enumerate(purzio_crystal.molecule.components): line 4 of 14
        print(f'Combination_id {comb_id}, Component_id {comp_id}') line 5
of 14
        analysed_mol = engine.analyse_molecule(component) line 6 of 14
        unusual_tors = [t for t in analysed_mol.analysed_torsions
                        if t.enough_hits and t.unusual] line 7 of 14
        if len(unusual_tors) > 0: line 8 of 14
            for t in unusual_tors: line 9 of 14
                print(f'Unusual torsion: {" ".join(t.atom_labels)}' line
10 of 14
                        f'Torsion: {t.value:.3f} °') line 11 of 14
            else: line 12 of 14
                print('No unusual torsions with enough hits.') line 13 of 14
        print() line 14 of 14
```

Lines 1 and 2 of this code block are just to provide some separation between the output of Parts 1 and 2, and this extension.

The final `print()` just adds some space to the output of this section. You can omit it if you prefer.

Save the changes and run the script. You should see additional output as shown on the next page.

Now using disorder combinations

Combination_id 0, Component_id 0

Unusual torsion: C27,C28,C29,C30 Torsion: -35.666 °

Unusual torsion: C39,C28,C29,C30 Torsion: 147.166 °

Combination_id 0, Component_id 1

No unusual torsions with enough hits.

Combination_id 1, Component_id 0

Unusual torsion: C45,C46,C29,C30 Torsion: -66.501 °

Unusual torsion: C47,C46,C29,C30 Torsion: 107.359 °

Unusual torsion: C38,C42,C43,C44 Torsion: 165.756 °

Unusual torsion: C41,C42,C43,C44 Torsion: -35.769 °

Unusual torsion: O7,C42,C43,C44 Torsion: 33.008 °

Combination_id 1, Component_id 1

No unusual torsions with enough hits.

You can download the full Python script from the ccdc-opensource GitHub repository [here](#).

Conclusion

In this example, we have seen how disorder is handled by the CSD Python API and we have specifically explored the disorder in the CSD entry PURZIO, which we analysed in Mercury in Example 1. We have seen that because of the way the disorder is described by assemblies and groups, the combinations can also be handled easily. Essentially the same geometry analysis that Mogul does is exposed via the Conformer API GeometryAnalyser class and you can tailor the output for your specific purposes.

```

1  #! /usr/bin/env python
2  #####
3  #
4  # This script can be used for any purpose without limitation subject to the
5  # conditions at http://www.ccdc.cam.ac.uk/Community/Pages/Licences/v2.aspx
6  #
7  # This permission notice and the following statement of attribution must be
8  # included in all copies or substantial portions of this script.
9  #
10 # 2025-09-29: created by the Cambridge Crystallographic Data Centre
11 #
12 #####
13
14 from ccdc.io import EntryReader
15 from ccdc.conformer import GeometryAnalyser
16 import matplotlib.pyplot as plt
17
18 # Read CSD entry
19 entry_reader = EntryReader('CSD')
20 purzio_entry = entry_reader.entry('PURZIO')
21 print(f'Refcode: {purzio_entry.identifier}')
22
23 # Get crystal
24 purzio_crystal = purzio_entry.crystal
25
26 # Examine disorder groups
27 print(f'Crystal has disorder: {purzio_crystal.has_disorder}')
28 print(f'Number of disorder assemblies: '
29       f'{len(purzio_crystal.disorder.assemblies)}')
30 for i, assembly in enumerate(purzio_crystal.disorder.assemblies, start=1):
31     print(f'Assembly {i} has {len(assembly.groups)} disorder groups.')
32
33 # Activate disorder assembly, group 1 of 2
34 first_assembly = purzio_crystal.disorder.assemblies[0]
35 first_group = first_assembly.groups[0]
36 first_group.activate()
37
38 print(f'Activated assembly_id: {first_assembly.id}')
39 print(f'Activated group_id: {first_assembly.active.id}')
40 print(f'Occupancy of disorder group: {first_group.occupancy:.3f}')
41
42 # Examine the molecules
43 mol = purzio_crystal.molecule
44 print(f'Number of molecules in crystal: '
45       f'{len(mol.components)}')
46
47 # Select component by index
48 comp_index = 0
49
50 # Examine the first molecule
51 comp = mol.components[comp_index]
52 print(f'Analysing molecule {comp_index + 1} of '
53       f'{len(mol.components)}')
54
55 # Check if this molecule is disordered
56 if any([atom.occupancy < 1 for atom in comp.atoms]):
57     print('This molecule is disordered.')
58 else:
59     print('This molecule is not disordered.')
60
61 # Identify disordered atoms
62 print([(atom.label, atom.occupancy) for atom in comp.atoms
63       if atom.occupancy < 1])
64
65 # Create an instance of geometry analyser
66 engine = GeometryAnalyser()
67
68 # Adjust settings - exclude organometallics and powder structures
69 engine.settings.organometallic_filter = 'organics_only'
70 engine.settings.powder_filter = True
71

```

```

71
72 # Analyse geometry
73 geom_analysed_mol = engine.analyse_molecule(comp)
74
75 # For example, look at torsion measurements
76 for torsion in geom_analysed_mol.analysed_torsions:
77     if torsion.unusual and torsion.enough_hits:
78         print(f'Unusual torsion: {"", ".join(torsion.atom_labels)} '
79               f'Torsion: {torsion.value:.3f} °')
80
81 # Accessing histogram data
82 # Take example of an unusual torsion
83 unusual_tors = [t for t in geom_analysed_mol.analysed_torsions
84                 if t.unusual and t.enough_hits]
85
86 # Retrieves the first occurrence of specified torsion
87 tor_query = next(t for t in unusual_tors if
88                  t.atom_labels == ['C39', 'C28', 'C29', 'C30'])
89
90 counts = tor_query.histogram()
91
92 # Create histogram
93 n_bins = len(counts)
94 bin_min, bin_max = 0, 180
95 bin_width = (bin_max - bin_min) / n_bins
96 bin_edges = [bin_min + i * bin_width for i in range(n_bins + 1)]
97
98 plt.figure(figsize=(8, 5))
99 plt.bar(
100     bin_edges[:-1],
101     counts,
102     width=bin_width,
103     edgecolor="black",
104     align="edge",
105     color="limegreen"
106 )
107
108 # Red line for torsion value
109 torsion_val = tor_query.value
110 plt.axvline(
111     torsion_val,
112     color="red",
113     linestyle="--",
114     linewidth=2
115 )
116
117 # Add text label above the histogram, slightly shifted
118 y_max = max(counts)
119 plt.text(
120     torsion_val + 2,      # shift slightly right
121     y_max * 0.9,         # place near the top
122     f"{torsion_val:.2f} °",
123     color="red",
124     fontsize=10,
125     rotation=90,
126     va="bottom"
127 )
128
129 plt.xlabel("Torsion Angle (degrees)")
130 plt.ylabel("Count of Observation Hits")
131 plt.title(f"Histogram for torsion {tor_query.atom_labels}")
132 plt.xlim(bin_min, bin_max)
133
134 plt.show()
135

```

```
135
136 print('\n-----\n')
137 print('Now using disorder combinations\n')
138
139 # Iterate over disorder combinations
140 for comb_id, combination in enumerate(purzio_crystal.disorder.combinations):
141     for comp_id, component in enumerate(purzio_crystal.molecule.components):
142         print(f'Combination_id {comb_id}, Component_id {comp_id}')
143         analysed_mol = engine.analyse_molecule(component)
144         unusual_tors = [t for t in analysed_mol.analysed_torsions
145                         if t.enough_hits and t.unusual]
146         if len(unusual_tors) > 0:
147             for t in unusual_tors:
148                 print(f'Unusual torsion: {",".join(t.atom_labels)}'
149                       f' Torsion: {t.value:.3f} °')
150         else:
151             print('No unusual torsions with enough hits.')
152     print()
153
```

Summary

In this workshop, you learned about how disorder is represented in the CSD and handled in analysis. You should now be able to:

- Use the disorder selector tool in Mercury
- Run a Mogul Geometry Check on disordered structures
- Select disorder assemblies and groups in the CSD Python API
- Iterate over disorder combinations
- Run geometry analysis in the CSD Python API

For your reference, you can find the Mercury and Mogul user guides [here](#) and the CSD Python API documentation [here](#).

Next Steps

If you have enjoyed this workshop, you might like to explore some more application of Mogul from the CSD-Core self-guided workshops [here](#). If you would like to explore the CSD Python API further, you could try the geometry analysis cookbook examples [here](#).

Feedback

We hope this workshop improved your understanding of Mogul and the CSD Python API as applied to disordered structures, and you found it useful for your work. As we aim to continuously improve our training materials, we would love to hear your feedback. Follow [the link](#) on the workshop homepage and insert the workshop code, which for this self-guided workshop is MOG-003. It will only take 5 minutes and your feedback is anonymous. Thank you!

Glossary

API

Application Programming Interface. This is a software intermediary that allows two applications to communicate.

Asymmetric unit

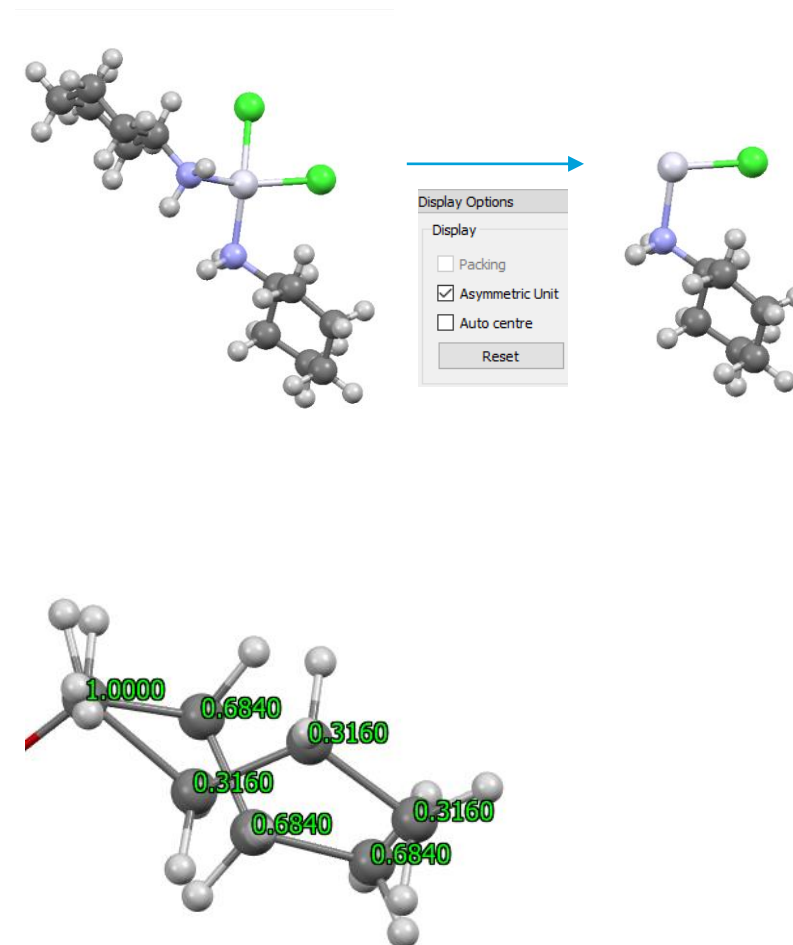
The asymmetric unit is the smallest part of a crystal structure from which the complete structure can be built using space group symmetry. The asymmetric unit may consist of only one molecule or ion, part of a molecule, or several molecules that are not related by crystallographic symmetry. (This is why you sometimes see more than one molecule in Mercury when you first display a structure from the Cambridge Structural Database.)

Deposited data

Data supplied to the CCDC from depositors (researchers, crystallographers) is in Crystallographic Information File (CIF) format and contains information about any disorder modelled by the crystallographer. CIFs are processed into a different format to create a CSD entry. Structures added to the CSD prior to the advent of CIF may not have any disorder model available. Mercury handles both disorder in CIFs and in CSD entries (if the disorder model is available).

Occupancy

The fraction of an atom modelled at a particular location in crystal structure. For non-disordered atoms, this is always one. If the atoms of a part of a structure are disordered, their occupancies must sum to one. For example, a disorder assembly with two disorder groups might have atoms with occupancies 0.75 and 0.25; a disorder assembly with three groups might have atoms with occupancies 0.8, 0.1 and 0.1. In both of those examples, the group with the highest occupancy is the major occupancy group and the remainder are the minor occupancy groups. Some disorder assemblies may not have major/minor groups, e.g. 0.5 and 0.5, or 0.33, 0.33 and 0.33.



A disordered butyl group with occupancies of the carbon atoms show.

Basics of Mercury Visualization

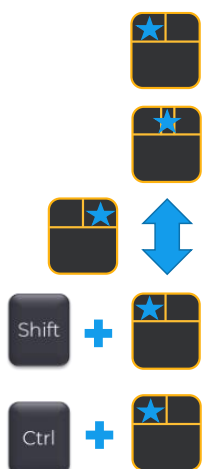
Mercury is the CCDC's visualization software to view 3D structures of small molecules, generate images, and animations of molecules.

In the following we will see some of the basics of navigation and visualization in Mercury that you will find helpful to support your analysis.

In the **Mercury interface** we find:

- **At the top:** list of menus from which we can access visualization and analysis options, and other CSD components such as CSD-Materials.
- **On the right-hand side:** the **Structure Navigator**, with the database loaded (depending on your licence). The Structure Navigator allows you to select a refcode to visualize in the main Mercury window.
- **Beneath the main display window:** **Display options toolbar**. You can quickly view a packing diagram, display Hydrogen bonding and detailed information about the molecule using the More Info option.

Using the **mouse to enhance visualization:**

- 
- Left mouse button and move – rotate molecules.
 - Middle Mouse wheel – move molecules up and down.
 - Right mouse button and move up and down – zoom in and out of molecules.
 - Shift + Left mouse button and move - rotate in the plane molecules.
 - Ctrl + Left mouse button and move - translate molecules.

Right click:

- Near a molecule and
- Away from a molecule

