Blender-Addon-Photgrammetry-Importer

Release 2.0.0

Sebastian Bullinger

Aug 31, 2023

CONTENTS

This documentation describes an addon for Blender that allows to import different reconstruction results of several `Structure from Motion` and `Multi-View Stereo` libraries.

Supported libraries (data formats):

- Colmap (Model folders (BIN and TXT), workspaces, NVM, PLY)
- Meshroom (MG, JSON, SfM, PLY)
- MVE (MVE workspaces) [1]
- Open3D (JSON, LOG, PLY) [1]
- OpenSfM (JSON)
- OpenMVG (JSON, NVM, PLY) [2]
- Regard3D (OpenMVG JSON)
- VisualSFM (NVM) [1]

In addition, the addon supports some common point cloud data formats:

- Polygon files (PLY) [3]
- Point Cloud Library files (PCD) [3]
- LASer files (LAS) [3, 4]
- LASzip files (LAZ) [3, 4, 5]
- Simple ASCII point files (ASC, PTS, CSV) [3]

[1] Requires `pillow` to read image sizes from disk. [2] Requires `pillow` for point color computation.
[3] Requires `pyntcloud` for parsing. [4] Requires `laspy` for parsing. [5] Requires `lazrs` for parsing.

The latest release of the addon is currently compatible with Blender 3.1.2 onwards. For older Blender versions you might find a suitable release here.

CHAPTER

ONE

GETTING STARTED

1.1 Installation Instructions

1.1.1 Delete any Previous Version of the Addon

- **Remove any previous version of the addon from Blender.**
  - Inside Blender go to `Edit/Preferences/Add-ons`, search for `Import-Export: Photogrammetry Import Export Addon` and click on `Remove`
  - See the *troubleshooting page* for more information.
- **THEN, CLOSE BLENDER**
- Reopen Blender and follow the installation instructions below

Without removal of previous versions errors may appear during activation or Blender may not reflect the latest changes of the addon.

1.1.2 Download the Addon for Blender 2.80 (or newer)

Option 1: Download a Release Version of the Addon

Download the corresponding `photogrammetry_importer.zip` from the release page.

Option 2: Download the Latest Version of the Addon

For example, clone the addon with

```
git clone https://github.com/SBCV/Blender-Addon-Photogrammetry-Importer.git
```

(Alternatively, go to `https://github.com/SBCV/Blender-Addon-Photogrammetry-Importer`, click on `clone or download`, and download the archive by clicking on `Download Zip`. Extract the `Blender-Addon-Photogrammetry-Importer-master.zip` file, which creates a folder `Blender-Addon-Photogrammetry-Importer`.)

Finally, compress the folder `photogrammetry_importer` in `Blender-Addon-Photogrammetry-Importer` to a zip archive `photogrammetry_importer.zip`. The final structure must look as follows:

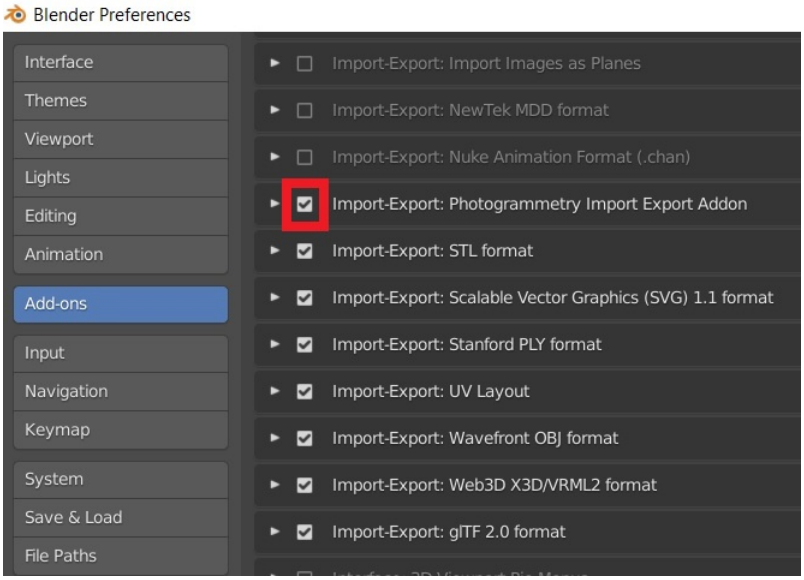
```
photogrammetry_importer.zip /
    photogrammetry_importer /
        blender_utility
        ext
        file_handler
        ...
        __init__.py
```

For convenience the repository contains a script (`create_blender_addon_zip.sh` for Linux, `create_blender_addon_zip.bat` for Windows) that creates the required `photogrammetry_importer.zip` file. Run the script without parameters (e.g. `./create_blender_addon_zip.sh`).

1.1.3 Install the Addon

Install the addon by

- Opening the preferences of Blender (`Edit / Preferences ...`)
- Select `Add-ons` in the left toolbar
- Click on `Install...` in the top toolbar
- Navigate to the `photogrammetry_importer.zip` file, select it and click on `Install Add-on`
- Scroll down to **ACTIVATE the addon**, i.e. check the bounding box left of `Import-Export: Photogrammetry Import Export Addon` (see image below)



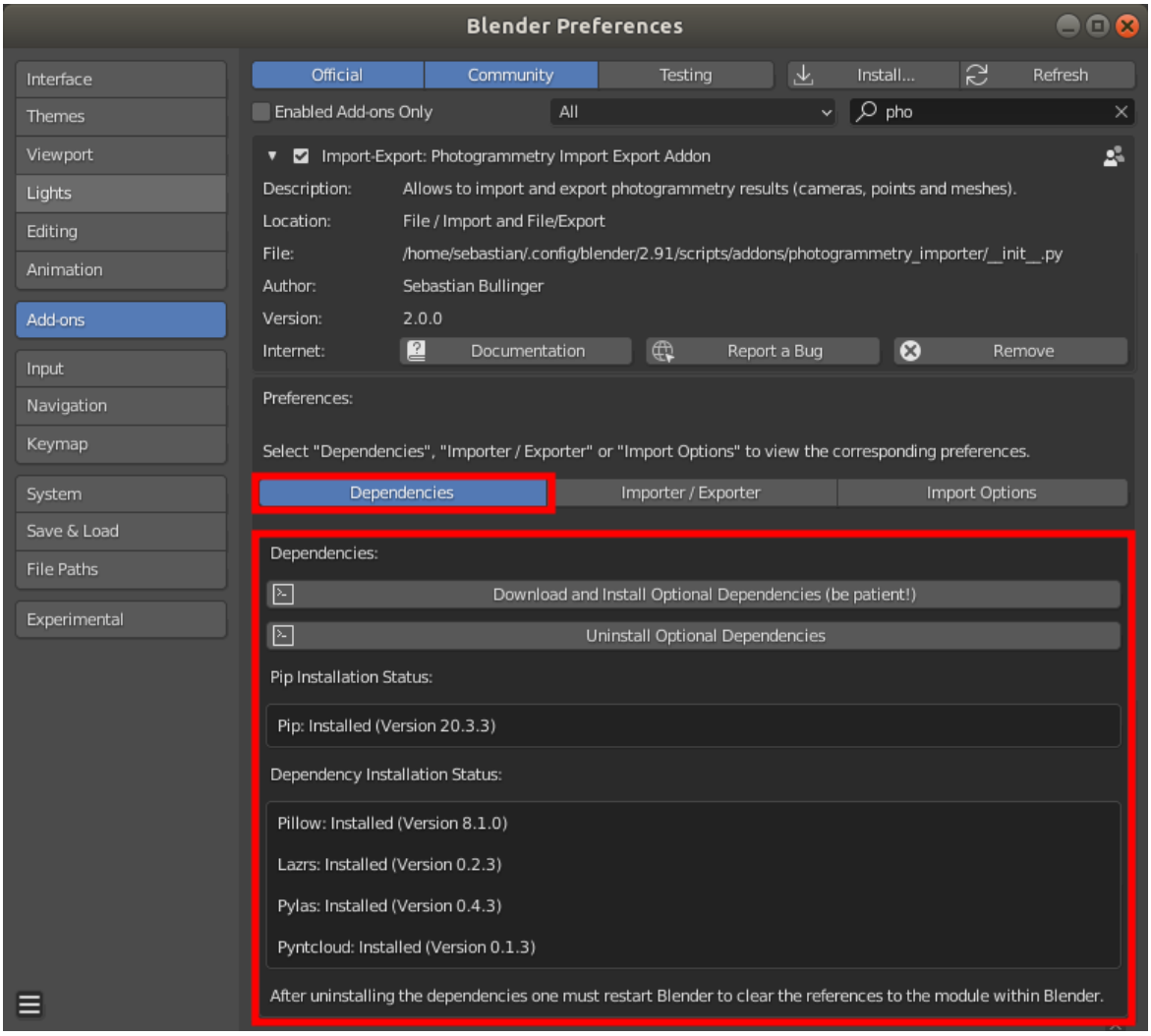
Follow the instructions on the *customize* page, to adjust the default options of the addon.

1.1.4 Install Optional Dependencies

This addon uses Pillow to read the (missing) image sizes from disk - required by the MVE, the Open3D and the VisualSFM importer. Pillow is also used to compute the (missing) point colors for OpenMVG JSON files. Using Pillow instead of Blender's image API significantly improves processing time. Furthermore, this addon uses Pyntcloud to import several point cloud formats such as `.ply`, `.pcd`, `.las`, `.laz`, `.asc`, `.pts` and `.csv`. For parsing `.las` and `.laz` files Laspy 2.0 (or newer), Lazrs and `Pyntcloud 0.3` (or newer) is required.

Option 1: Installation using the GUI (recommended)

Requires Blender 2.83.5 or newer. **Administrator privileges might be required to install the dependencies (depending on the location of the Blender installation directory).**



Clicking on `Download and Install Optional Dependencies` installs `pip` (if not already present) and uses the `pip` executable to install the actual dependencies. Start `Blender` from the command line to see the installation progress and potential error messages.

Note: If you experience problems while *updating* the dependencies, try a fresh Blender installation.

Note: If you use VSCode with Blender VSCode to run this addon, the installation of laspy will fail. In this case you need to install it manually (see below).

Option 2: Installation using the command line

In case the installation using the GUI does not work, it is possible to install the dependencies with the command line.

If you haven't installed pip for Blender already, download https://bootstrap.pypa.io/get-pip.py and copy the file to

```
<Blender_Root>/<Version>/python/bin
```

For Linux run:

```
<Blender_Root>/<Version>/python/bin/python3.7m <Blender_Root>/<Version>/python/bin/get-
↪pip.py
<Blender_Root>/<Version>/python/bin/pip install pillow
<Blender_Root>/<Version>/python/bin/pip install lazrs
<Blender_Root>/<Version>/python/bin/pip install laspy
<Blender_Root>/<Version>/python/bin/pip install pyntcloud
```

For Windows run:

```
<Blender_Root>/<Version>/python/bin/python.exe <Blender_Root>/<Version>/python/bin/get-
↪pip.py
<Blender_Root>/<Version>/python/Scripts/pip.exe install pillow
<Blender_Root>/<Version>/python/Scripts/pip.exe install lazrs
<Blender_Root>/<Version>/python/Scripts/pip.exe install laspy
<Blender_Root>/<Version>/python/Scripts/pip.exe install pyntcloud
```

IMPORTANT: Use the full path to the python and the pip executable. Otherwise the system python installation or the system pip executable may be used.

1.2 Troubleshooting

1.2.1 Known (Blender) Issues

Please see this issue for an up-to-date list of limitations.

1.2.2 Problems to Activate the Addon

If you experience problems while installing and activating a newer version of the addon (i.e. an older version of the addon was previously installed), delete Blender's user folder of the addon. This page of the Blender manual provides information about the location of the corresponding folder.

Make sure that you CLOSE Blender BEFORE deleting the folder.

Windows

Under Windows delete the following folder:

```
%USERPROFILE%\AppData\Roaming\Blender Foundation\Blender\<Version>\scripts\addons\
↪photogrammetry_importer
```

In the case of Blender 2.82:

```
%USERPROFILE%\AppData\Roaming\Blender Foundation\Blender\2.82\scripts\addons\
↪photogrammetry_importer
```

Linux

Under Linux delete:

```
~/.config/blender/<Version>/scripts/addons/photogrammetry_importer`
```

In the case of Blender 2.82:

```
~/.config/blender/2.82/scripts/addons/photogrammetry_importer
```

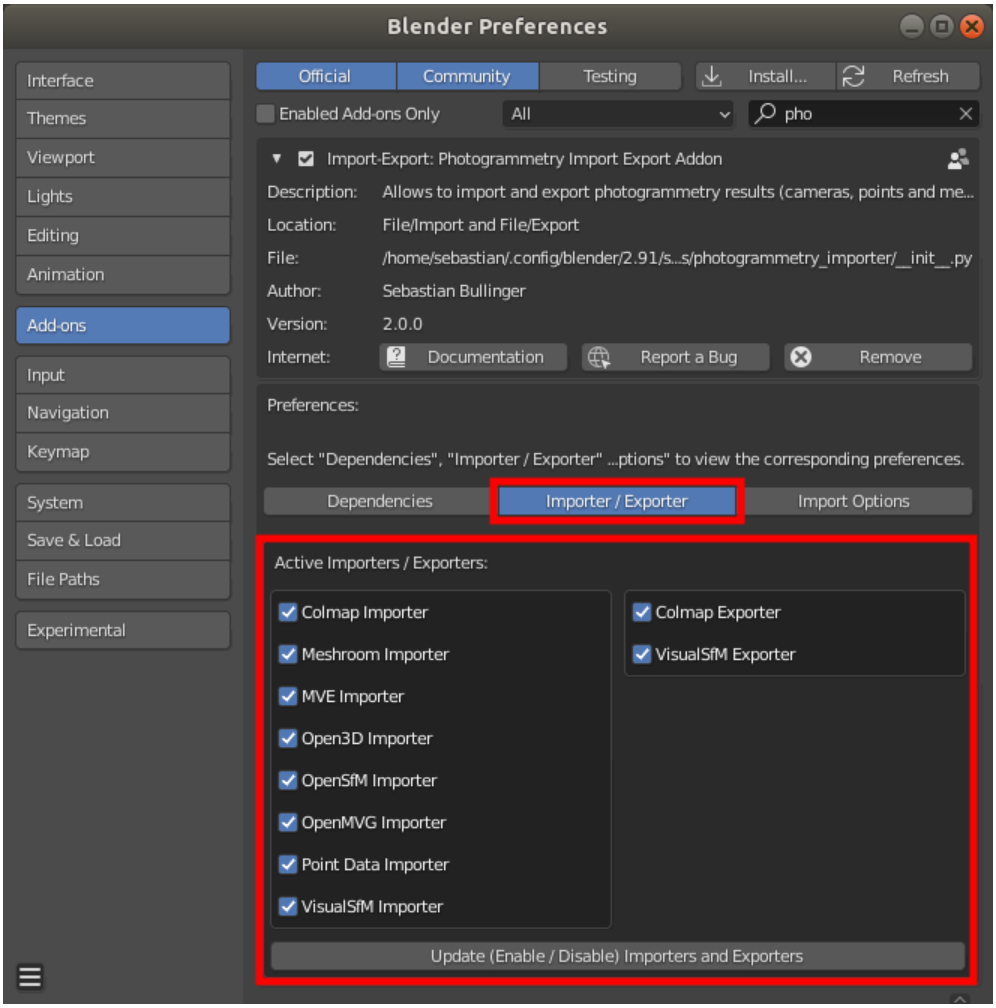
1.2.3 Blender Crashes while Importing Reconstructions

This is probably not an issue of the addon or Blender, but caused by outdated graphic drivers. If the problem persists (after restarting Blender / the operating system), one can find more information in the Blender manual. As workaround one may import the point cloud as Blender object or as a particle system (by adjusting the corresponding import options) instead of drawing the point cloud with OpenGL (which is the default import option).

1.3 Customize Import/Export Options

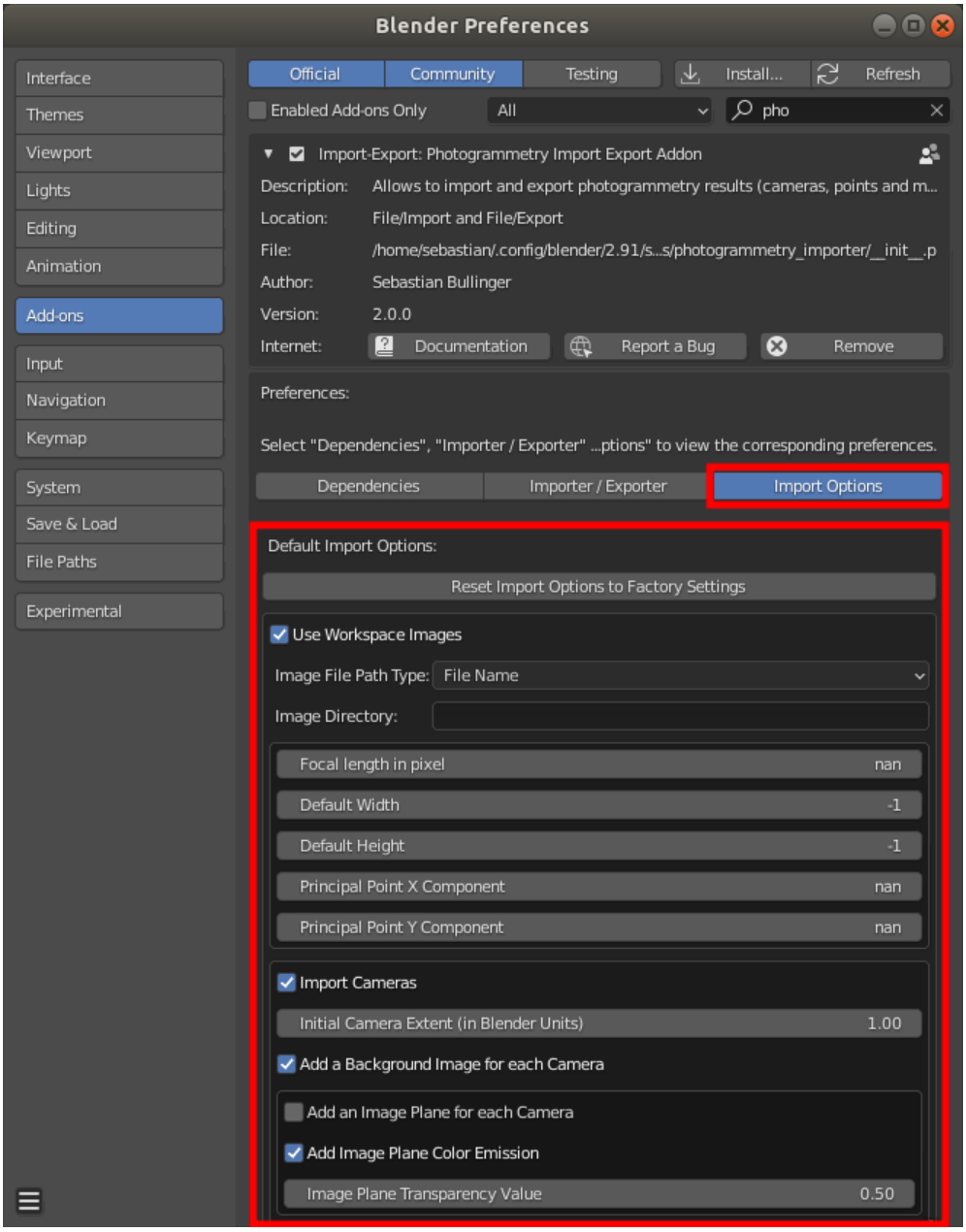
1.3.1 Enable / Disable Importers and Exporters

If you want to only add a subset of the provided import/export functions, adjust the checkboxes in the *Addon preferences* and click on the `Update (Enable/Disable) Importers and Exporters` button as shown below:



1.3.2 Adjusting Default Import Options

Furthermore, it is possible to set the default import options by adjusting the corresponding settings in the *Addon Preferences* - see the image below.



1.4 Examples

This repository contains under `Blender-Addon-Photogrammetry-Importer/examples` several reconstruction results of the following Structure from Motion libraries:

- Colmap (colmap_example_model_bin and colmap_example_model_txt)
- Meshroom (meshroom_example.json)
- OpenMVG (OpenMVG_example.json)
- VisualSfM (VisualSFM_example.nvm)

If you want to show the corresponding images in Blender, download the corresponding dataset and copy the images to

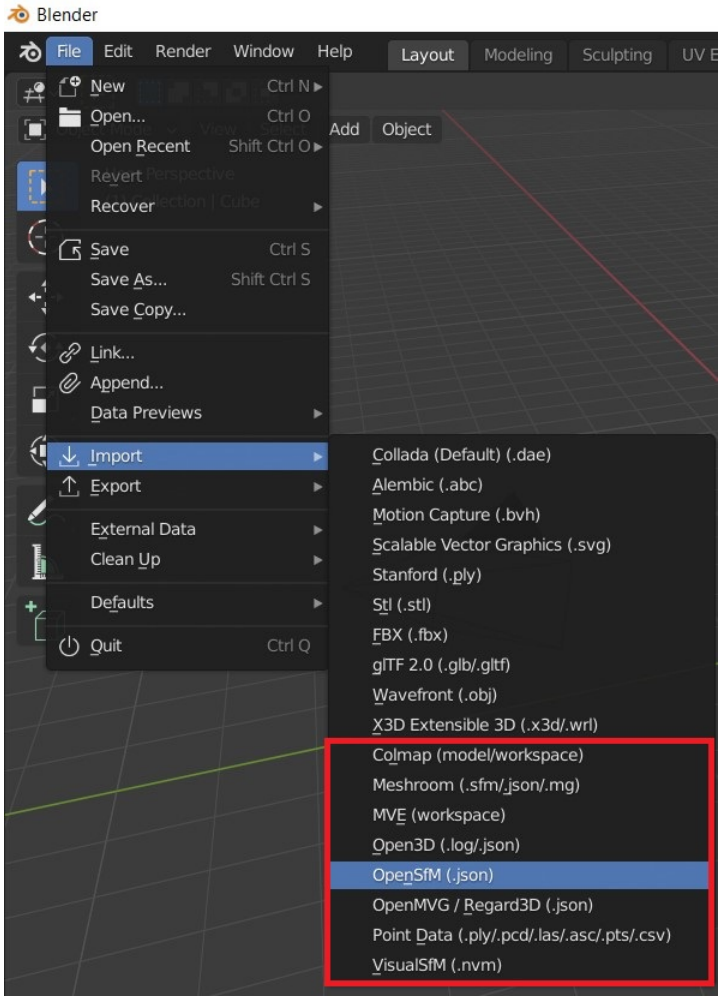
`Blender-Addon-Photogrammetry-Importer/examples/images`. Alternatively, you can adjust the corresponding path in the import options of the addon.

1.5 Import Data

Errors, help and logging information during import / export is shown in the `Info` area. Check this output, if nothing is imported. Probably the default width and height values are not set (see below).

1.5.1 General

In Blender use `File/Import/<Import Function>` to import the corresponding file.



For each camera one can add the corresponding image plane. Pillow is required to read the images from disc. Use the import dialog to adjust the `image path`. By default the addon searches for the images in the folder where the reconstruction file is located. **This addon uses the node system of Cycles to visualize the image planes. Thus, the addon switches automatically to Cycles, if image planes are added.**

There is an option to represent the point cloud with a particle system. This allows you to render the point cloud. A single texture is used to store the color of all particles. **The color of the points / textures of the images are visible, if "Cycles Render" is selected and the 3D view is set to "Material".** Eevee does not (yet) support `particle info`

nodes. (Checkout the manual for more information.) Thus, it is currently **not possible** to render point clouds with individual particle colors **in Eevee**.

1.5.2 VisualSfM

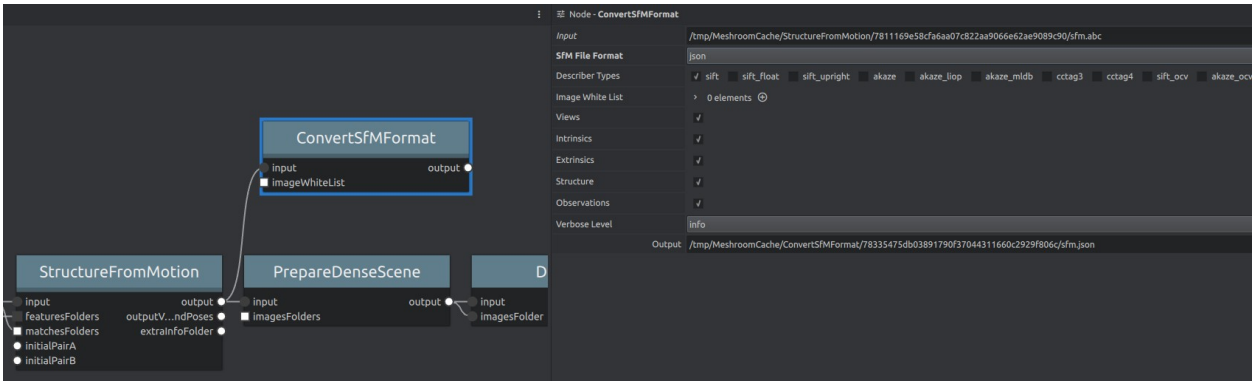
The addon automatically looks for the fixed calibration line in the NVM file (i.e. `NVM_V3 FixedK fx cx fy cy r` (first line)). Without the fixed calibration line the addon assumes that the principal point is at the image center. NVM files contain no information about the size of the images. Use the import dialog to adjust the `image path` to automatically read the image size from disc or set the default `width` and `height` values.

1.5.3 OpenMVG JSON

The OpenMVG `JSON` files contain no color information. If you want to import a point cloud with color information, you might want to use the `openMVG_main_ComputeSfM_DataColor` tool (see this link) and import the corresponding `*.ply` file.

1.5.4 Meshroom

By default Meshroom stores the Structure from Motion results (i.e. cameras and points) in Alembic (`*.abc`) files. Since parsing `*.abc` files requires building additional (heavy) dependencies, e.g. this library, it is currently not supported by this addon. However, one can add a `ConvertSfMFormat` node in Meshroom (see image below) to write the reconstruction result to `*.SfM` / `*.json` files.



In addition to `*.SfM` / `*.json` files the addon allows to import `*.mg` files, which allows to also import corresponding meshes. The addon prioritizes the output of recently added nodes (e.g. `ConvertSfMFormat3` has a higher priority than `ConvertSfMFormat`). For importing meshes the addon uses the following prioritization: first the output of `Texturing`, then the output of `Meshfiltering` and finally the output of `Meshing`. **After adding the nodes (e.g. `ConvertSfMFormat`), do not forget to save your project (i.e. the `*.mg` file),** since the addon uses this file to determine available reconstruction results.

In order to import the original images corresponding to the `*.mg` file, one can set the import option `Image File Path Type` of the Blender-Addon to `Absolute Path`. To import the undistorted `*.exr` images set `Image File Path Type` to `File Name` and set `Image Directory` to the folder with the `*.exr` files.

1.5.5 Regard3D

By default Regard3D stores the Structure from Motion results in `path/to/project/pictureset_0/matching_0/triangulation_0/sfm_data.bin`. Use OpenMVG to convert the `*.bin` to a `*.json` file with `openMVG_main_ConvertSfM_DataFormat -i path/to/sfm_data.bin -o path/to/cameras.json`. For Windows you can find the pre-built binaries of OpenMVG here.

1.5.6 ASCII

Each line in an ASCII file (`.asc/.pts/.csv`) represents a point with several attributes (coords, normals, colors, ...). In the case of `.asc/.pts/` there might be an optional header such as `//X Y Z Rf Gf Bf Intensity` or `//X Y Z Intensity R G B` that defines the order of the attributes. If no header is provided, the addon tries to estimate the order of the attributes. The color attributes can be defined as integer values (`R G B`) between `0` and `255` or float values (`Rf Gf Bf`) between `0.0` and `1.0`. Attributes other than position and color are ignored by the addon.

1.5.7 Meshes

In order to view a reconstructed mesh with the corresponding sparse reconstruction (cameras and point cloud) import the files separately. When importing `*.obj` files make sure to adjust the corresponding import transform options. Set the `Forward` option to `Y Forward` and the `Up` option to `Z Up`.

1.5.8 Limitations

Blender supports only global render settings (which define the ratio of all cameras). If the reconstruction file contains cameras with different aspect ratios, it is not possible to visualize the camera cones correctly. Furthermore, radial distortions of the camera model used to compute the reconstruction will result in small misalignment of the cameras and the particle system in Blender.

1.6 Export Data

Errors, help and logging information during export is shown in the "Info" area.

The addon allows to export camera poses and vertex positions to a few photogrammetry formats. Currently, the addon supports:

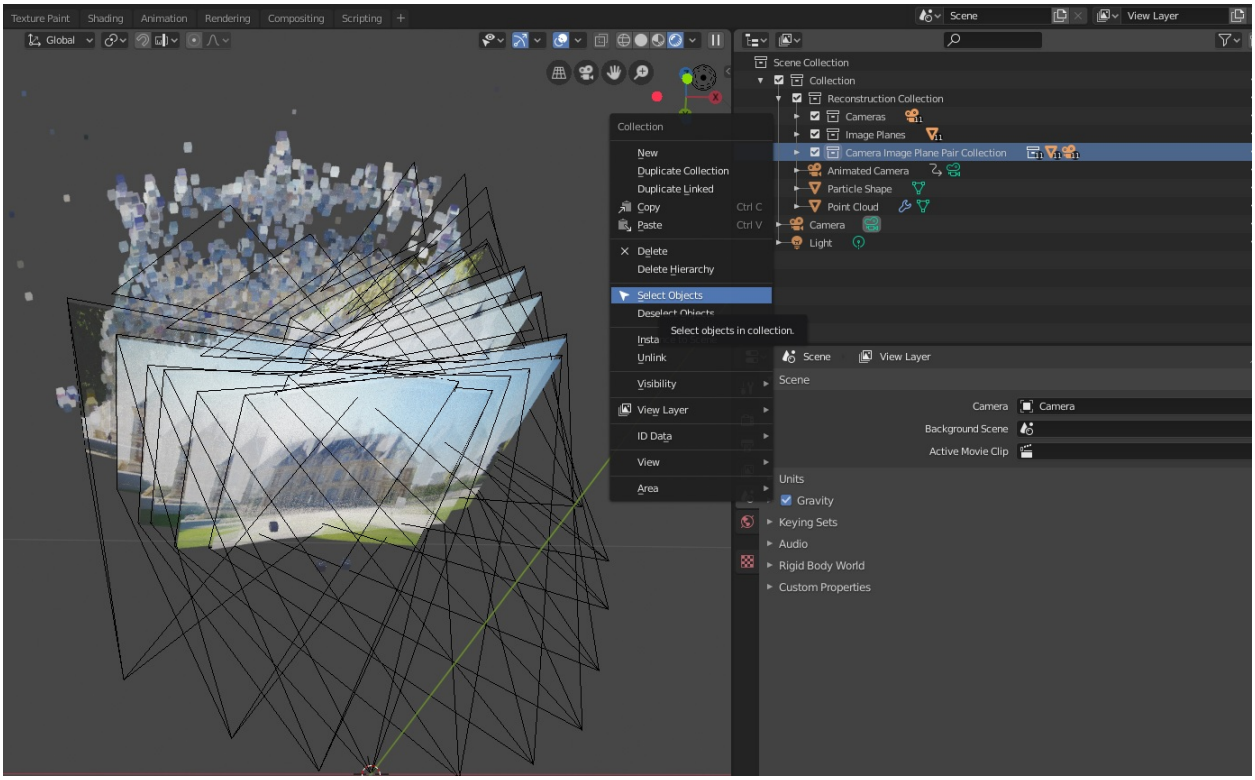
- Colmap model folders
- NVM files of VisualSFM

Select all cameras and objects you want to export. For each selected mesh the vertices are stored as points in the output file/folder. Use `File/Export/<Export Function>` to export the corresponding file.

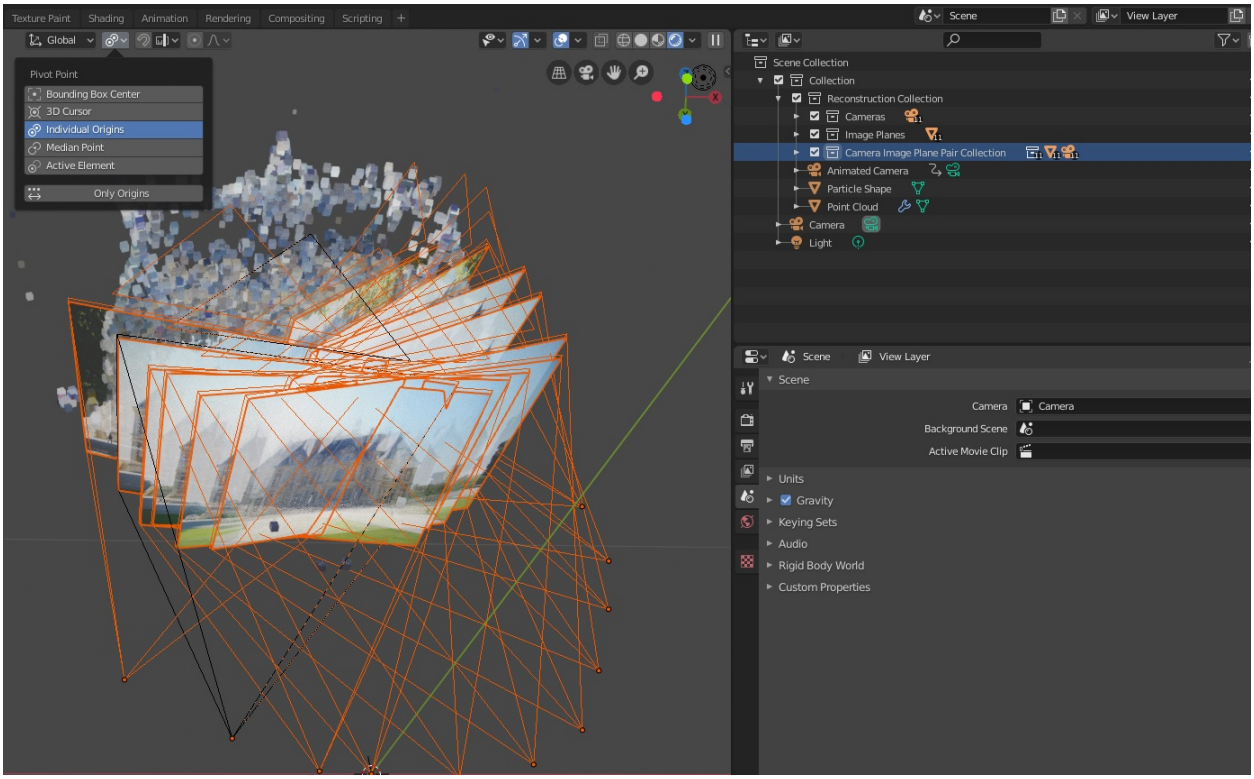
1.7 Scale Cameras and Points

1.7.1 Adjust the Scale of Cameras after Importing

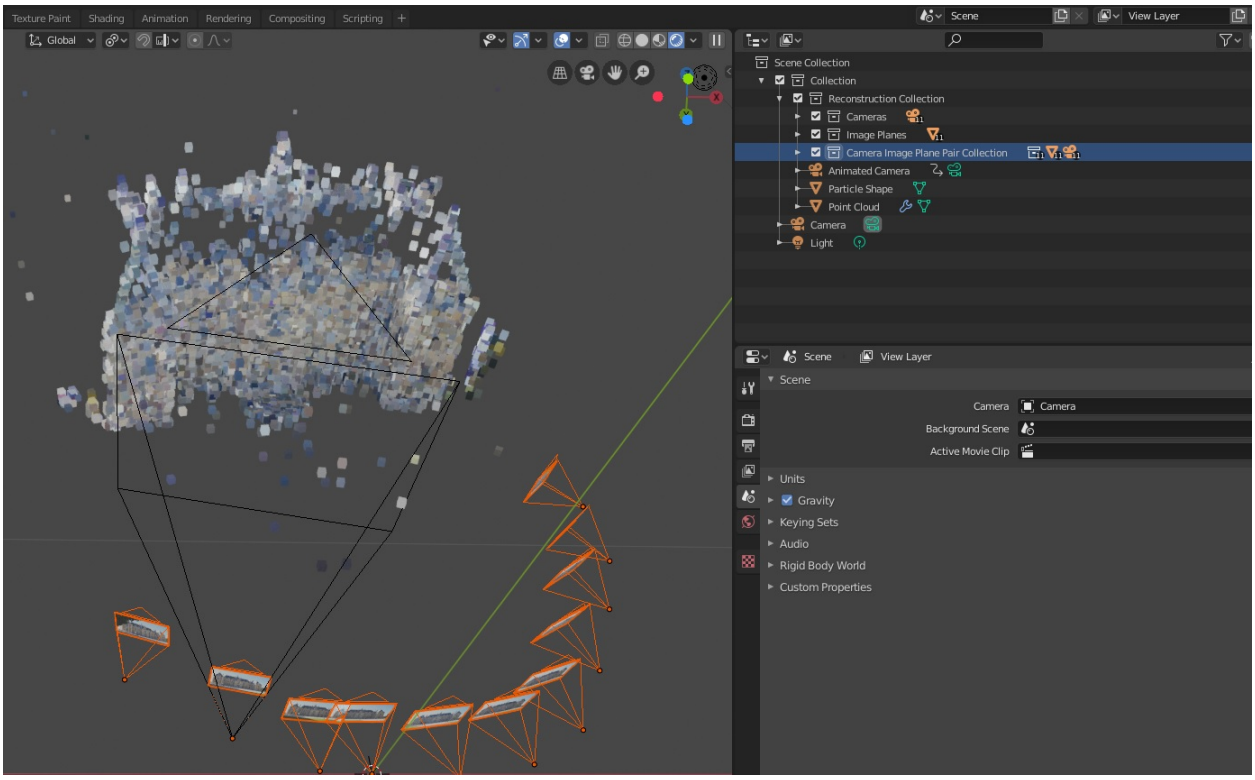
1.) Select the cameras and the corresponding image planes, i.e. right click on the collection `Camera Image Plane Pair Collection` and in the context menu on `Select Objects`.



2.) Select the individual origins as pivot points, i.e. click in the 3D view on `Pivot Point` and select `Individual Origins`.

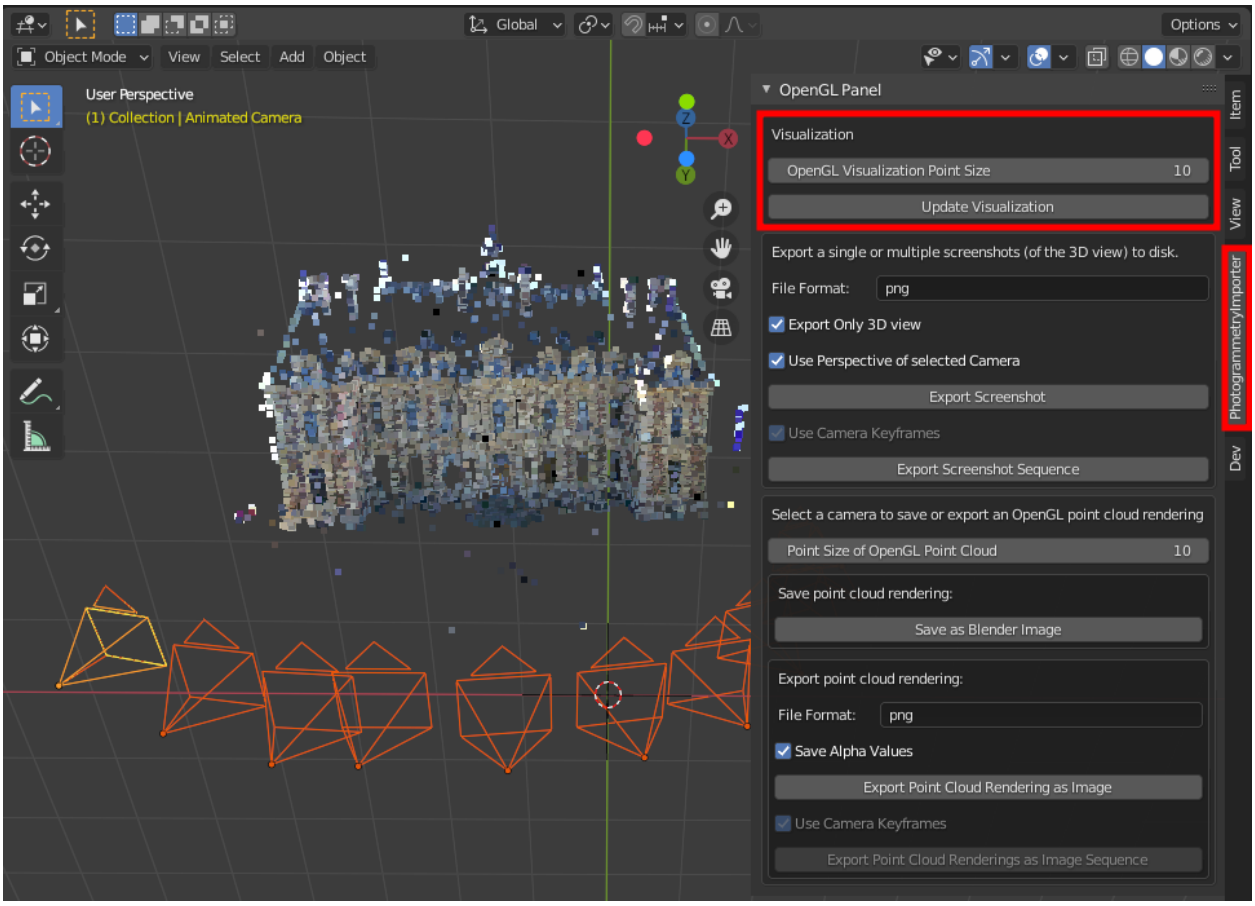


3.) Scale the objects, i.e. press `s` and move the mouse or press `s` and enter the scaling factor.



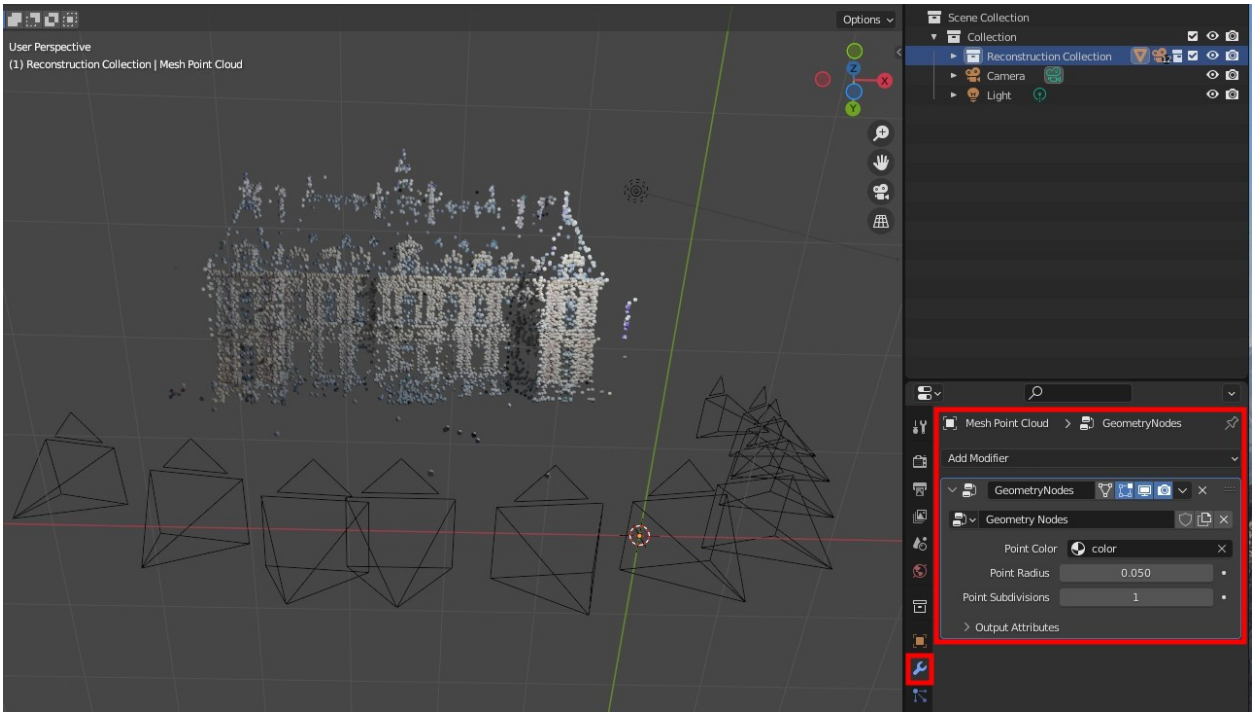
1.7.2 Adjust the Scale of OpenGL Points Drawn in the 3D View (After Importing)

The size of the points in the OpenGL point cloud can be defined using the panel in the 3D view.



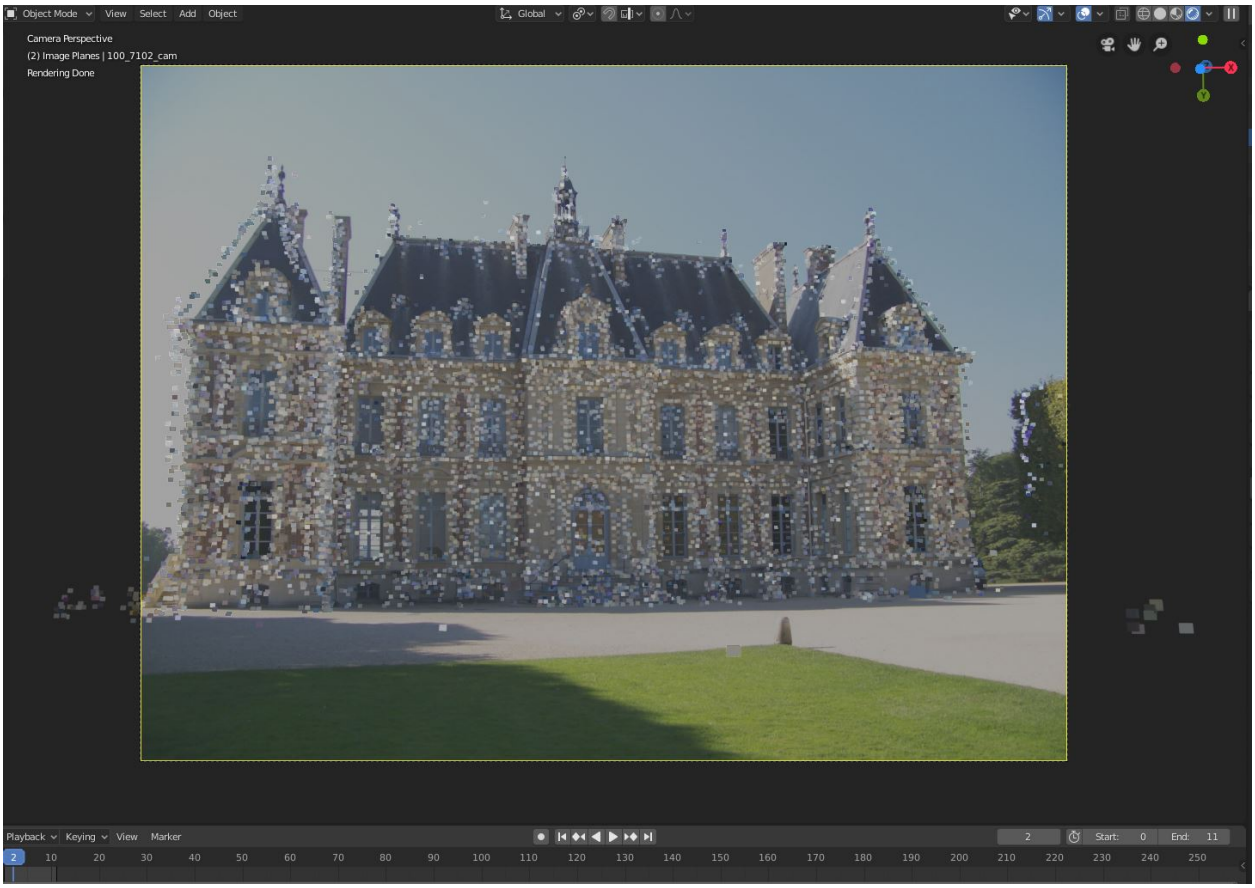
1.7.3 Adjust the Scale/Shape of Geometry Node Points of a Mesh Object (After Importing)

The input of the Geometry Nodes (i.e. `point radius` and the `point subdivision`) can be adjusted using the `Properties` editor (`Shift + F7`).



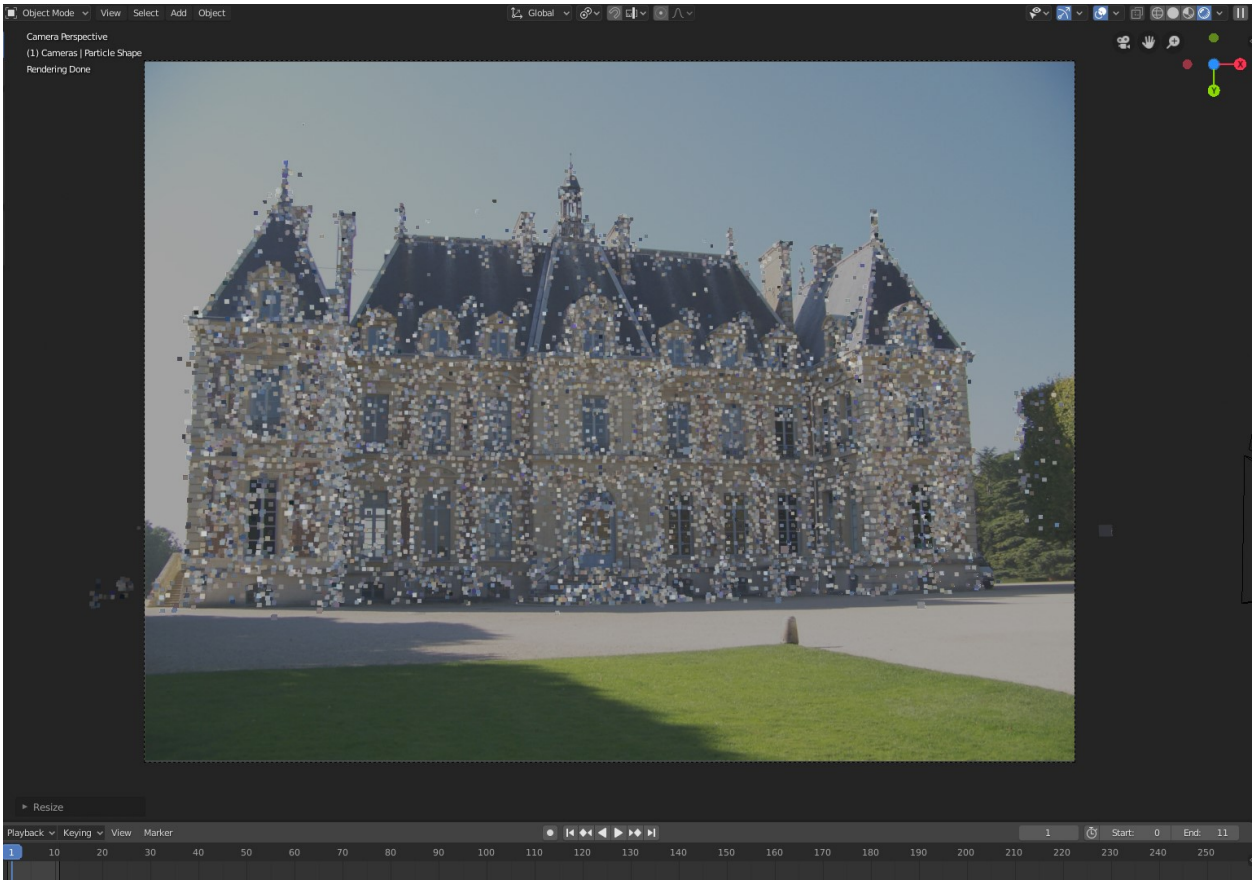
1.8 Alignment of Cameras and Point Cloud

The 3D viewport of Blender unfortunately does not support radial camera distortions. If the imported Structure from Motion reconstruction has a camera model with radial distortion, the image planes and background images will not perfectly align with the 3D point cloud. For example, such cases occur if the intrinsic camera parameters are optimized (not fixed) during the Structure from Motion reconstruction. The following image shows a corresponding example. For instance, see the offset between the points at the left and the right side of the building.

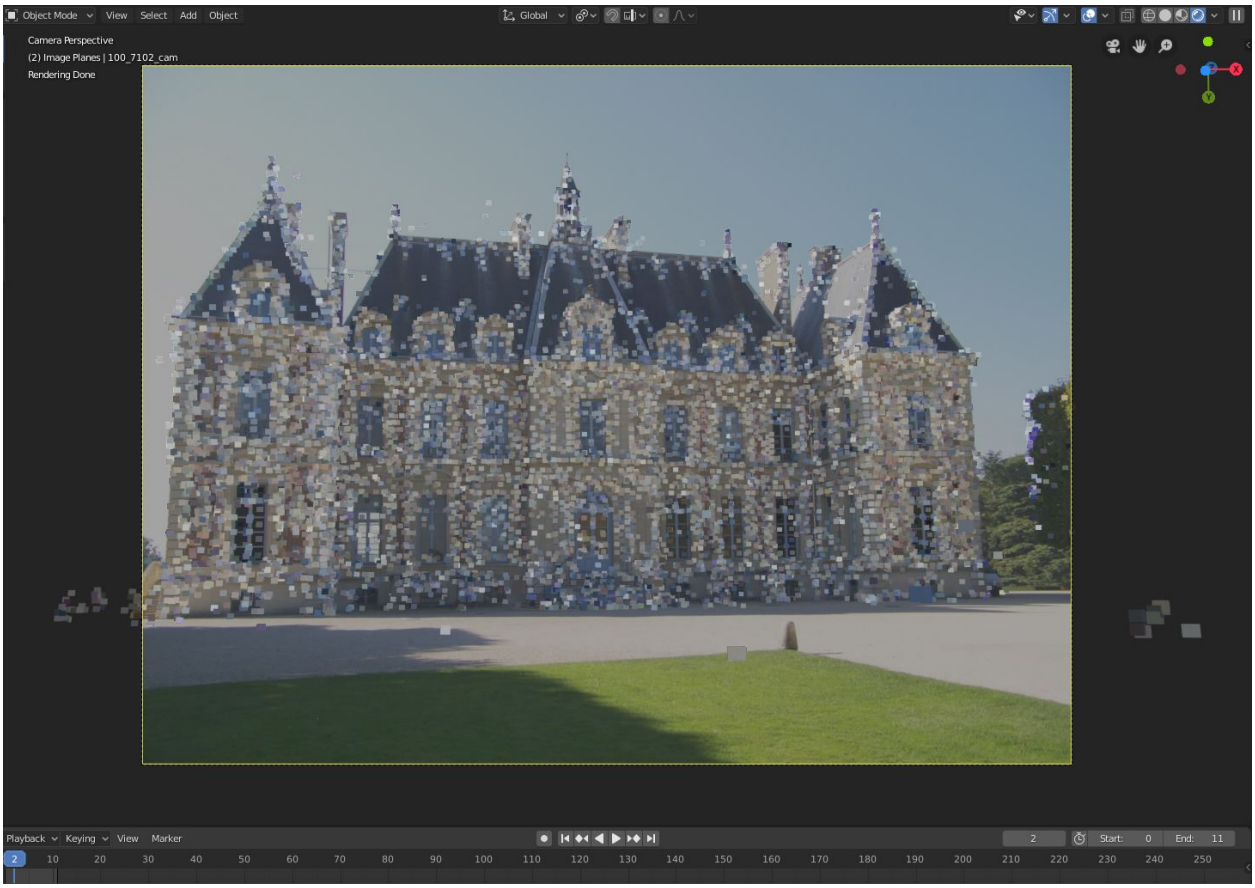


There are two ways to address this issue.

Option 1: This approach is only applicable, if the images are not subject to radial distortion. Provide the (true) intrinsic camera parameters and fix/lock these during reconstruction. The following image shows the corresponding result.



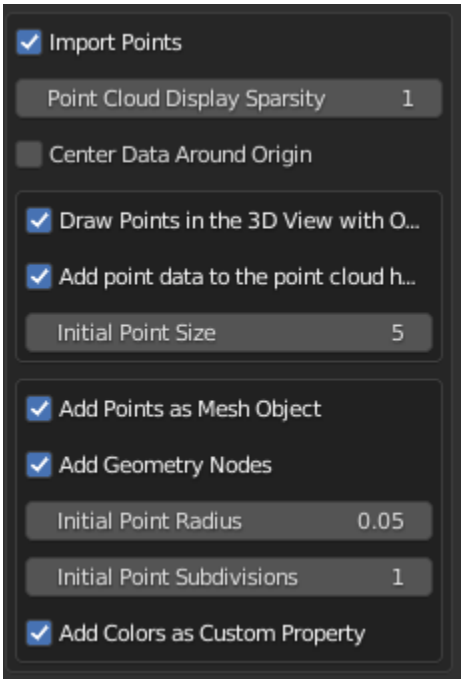
Option 2: After computing the Structure from Motion reconstruction perform the first step (i.e. image undistortion) of the dense reconstruction (Multi-View Stereo). Using the undistorted images instead of the original imagery resolves the alignment problem. For example, Colmap and Meshroom allow to compute undistorted images. An example is shown below.



1.9 Visualization and Rendering

Currently, this addon supports the following two point cloud visualization options:

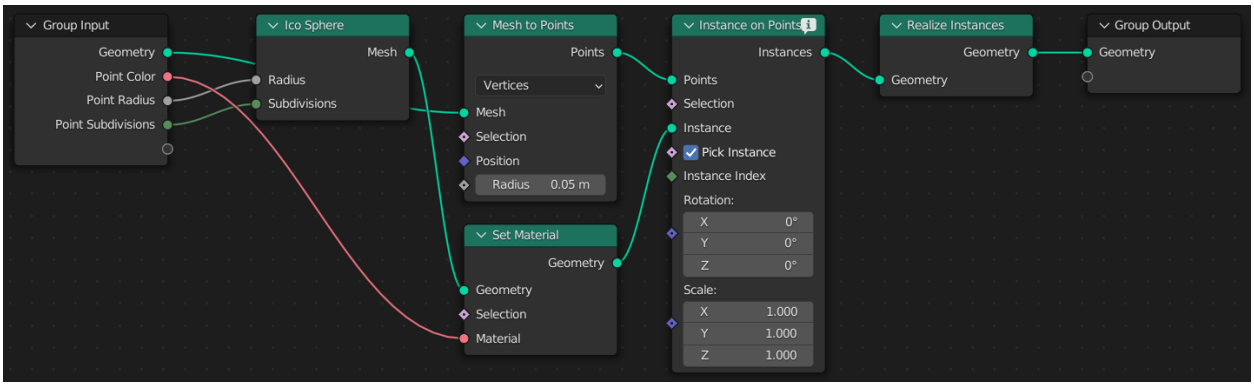
- Representing the points with a mesh object (using Blender's Geometry Nodes)
- Visualizing the points with OpenGL



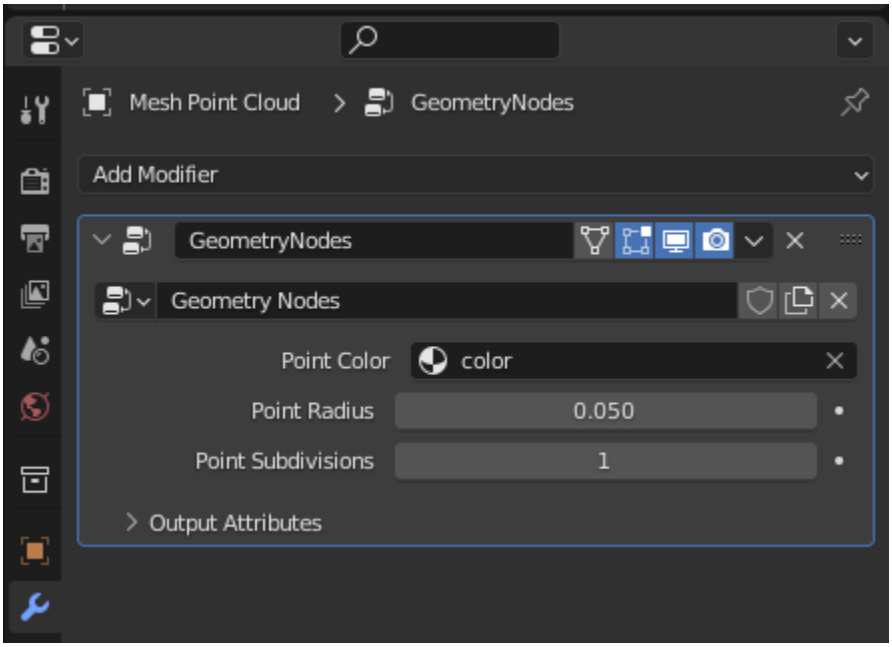
Each option has different advantages / disadvantages.

1.9.1 Option 1: Representing the Points with a Mesh Object (using Blender's Geometry Nodes)

If selected, the addon adds a blender object with a vertex for each point in the point cloud. The option `Add Geometry Nodes` create several geometry nodes (see image below) that allow to render the point cloud with Eevee or Cycles.



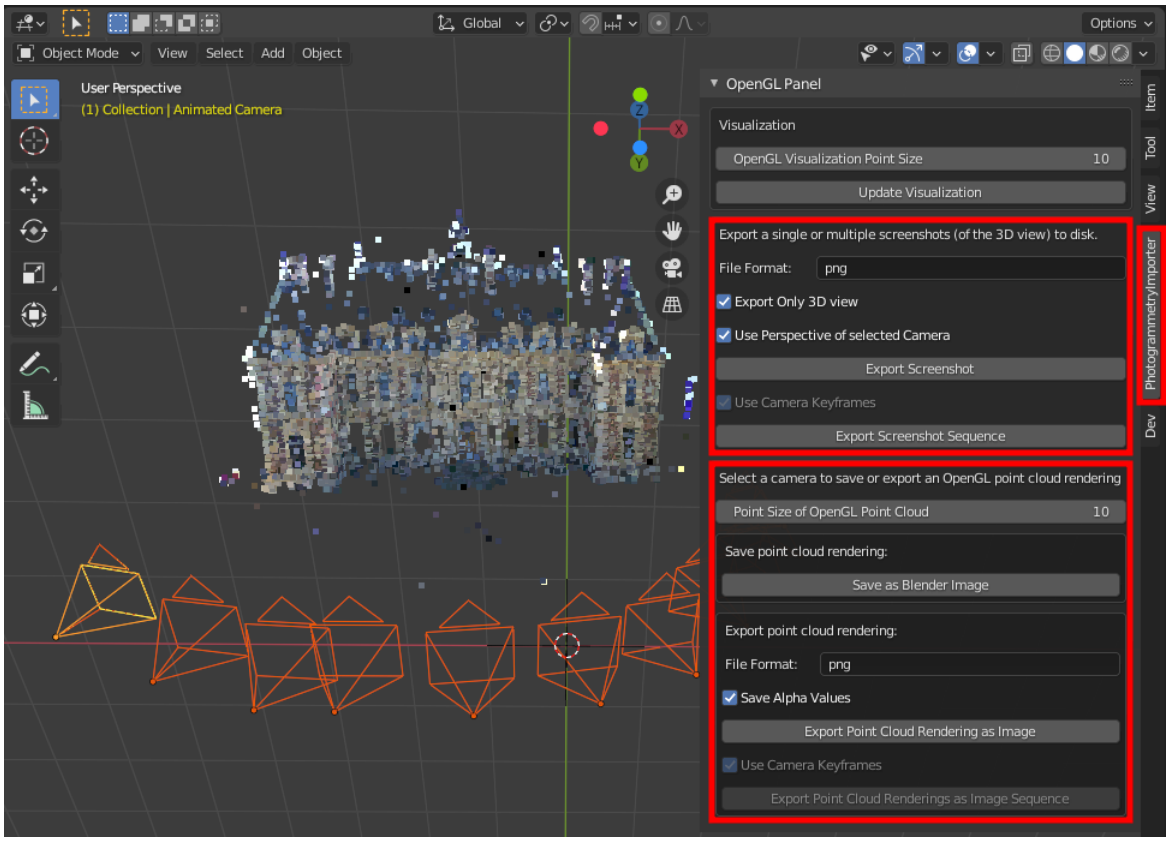
The `point radius` and the `point subdivision` can be adjusted after import using the `Properties editor` (`Shift + F7`).



1.9.2 Option 2: Visualizing and rendering the points with OpenGL

If selected, the point cloud is shown in the Viewport using Blender's OpenGL API. That means, there is **no** Blender object representing the points in the point cloud. The pose (i.e. rotation and translation) of the object can be changed by adjusting the corresponding "anchor" (i.e. a Blender `empty` object).

- Advantage: Low computational costs for visualization.
- Disadvantage: It is not possible to render these points with the render functions provided by Blender. However, this addon provides a panel to save/export OpenGL renderings of the points using an offscreen buffer or Blender's screenshot operator (see image below).



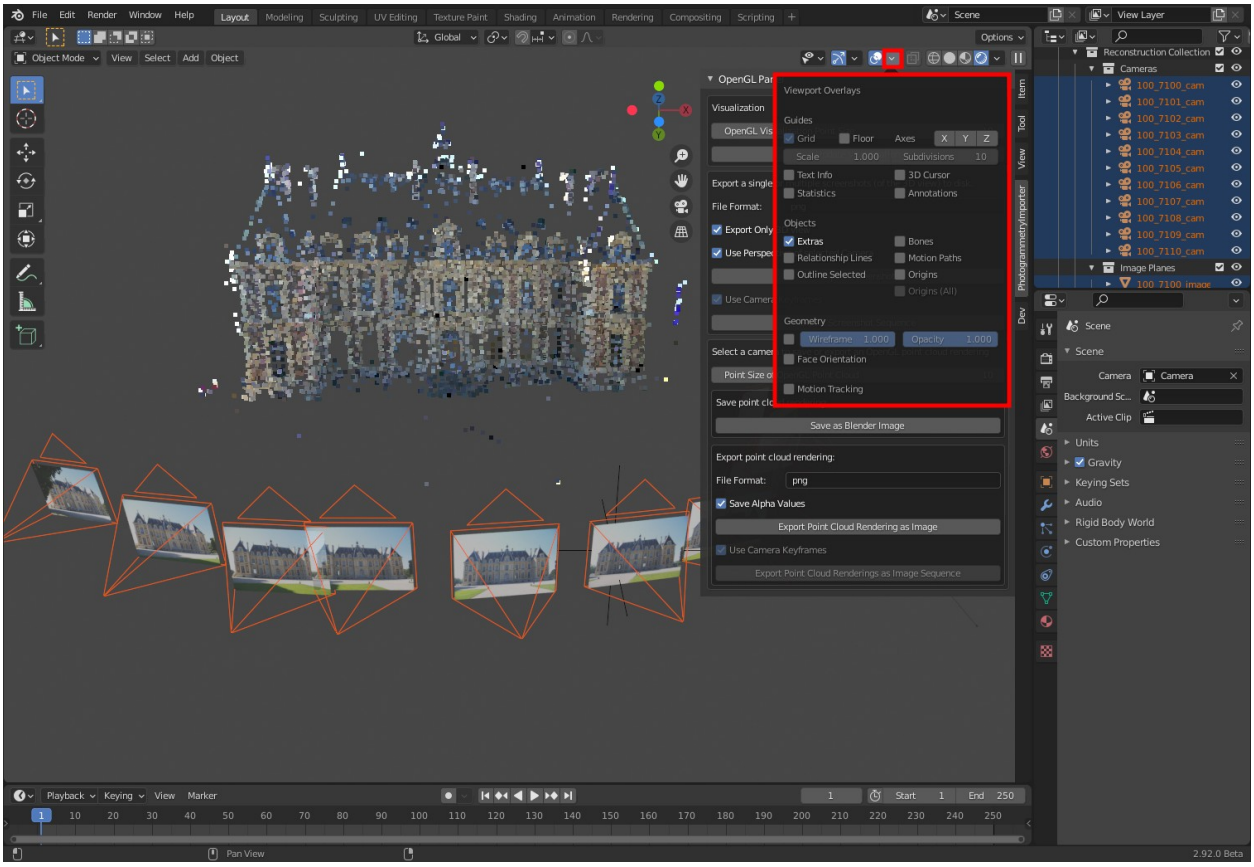
Option 2a: Write results to disk with Blender's offscreen buffer

Rendering the scene with Blender's offscreen buffer renders (only!) the points drawn with Blender's OpenGL API to disk. In order to render other elements such as cameras, image planes, meshes etc use Blender's screenshot operator - see below.

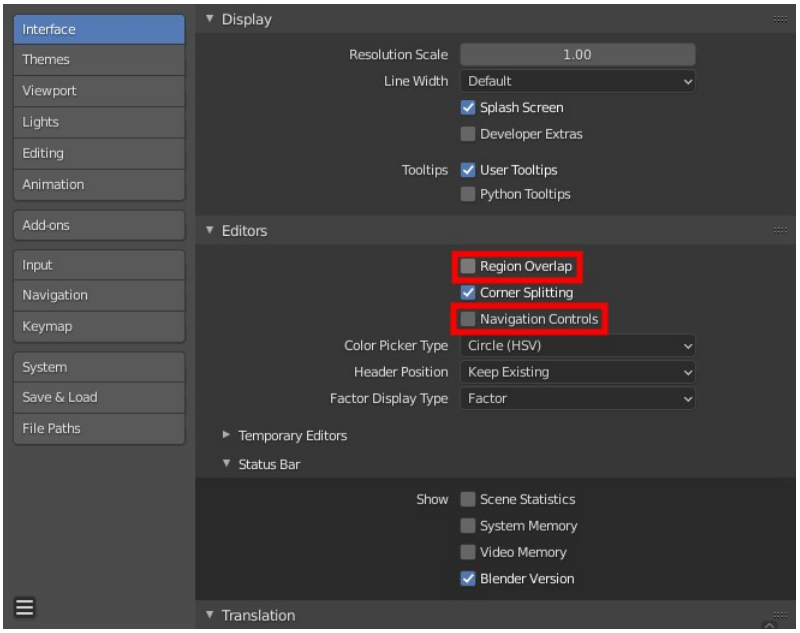
Option 2b: Write results to disk with Blender's screenshot operator

Since Blender's screenshot operator renders all visible elements of the viewport to disk it is usually convenient to adjust the appearance.

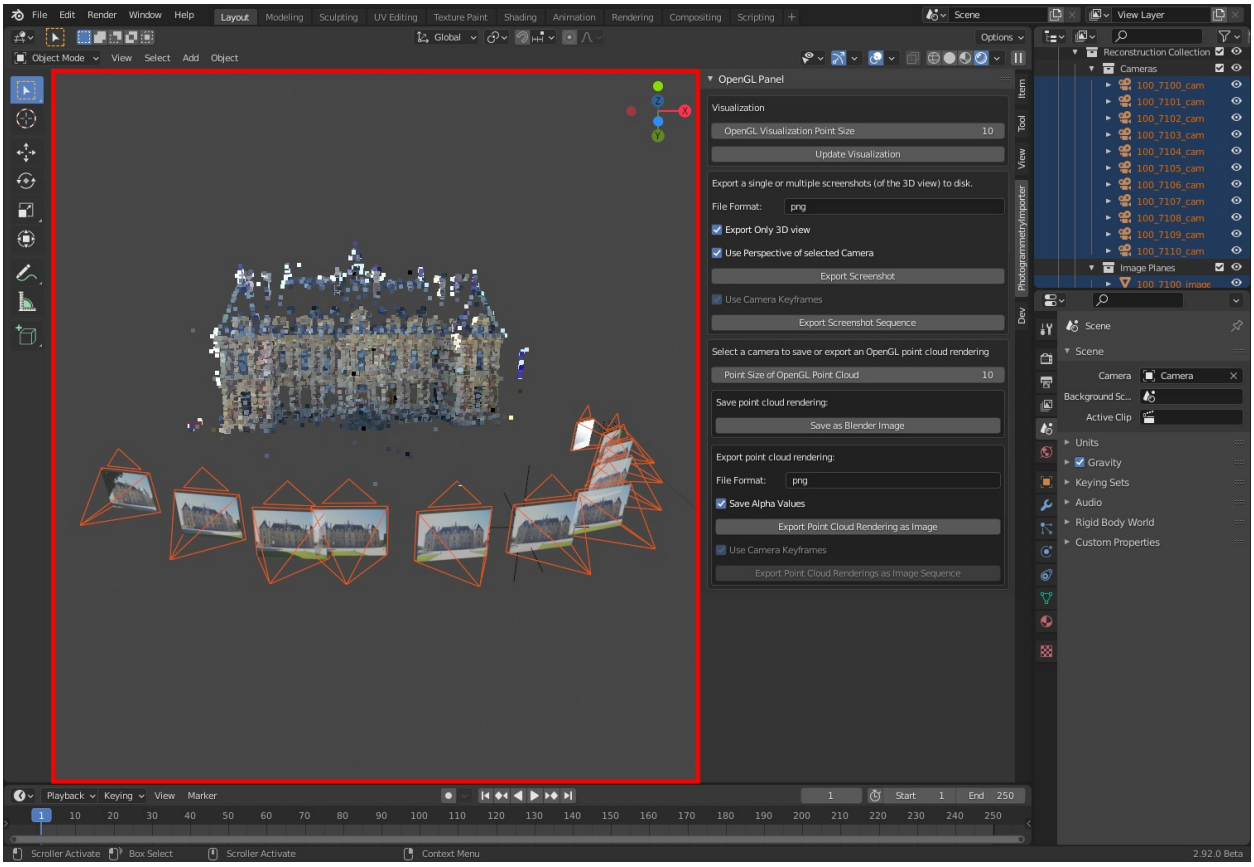
In order to hide gridlines, axes etc. click on the `Overlays` button in Blenders 3D viewport and disable the corresponding options - see the image below.



To ensure that the reconstruction results are not occluded by Blender panels go to `Edit / Preferences ...` and uncheck the option `Region Overlap` - as shown in the following image. There, it is also possible to hide the `Navigation Controls`.



After adjusting these options the viewport looks as follows.



1.10 Addon Usage with Python

There are two ways to access the functionality of the addon with Blender's Python console / text editor (after installation and activation of the addon):

- Import the addon as Python module
- Call the appropriate operator registered in bpy.ops.import_scene

1.10.1 Option 1: Import the addon as Python module

According to the documentation:

> *The only difference between addons and built-in Python modules is that addons must contain a bl_info variable*

Therefore, after installation and activation one can use Python's standard import syntax to import different classes and functions such as:

```
from photogrammetry_importer.types.camera import Camera
from photogrammetry_importer.file_handlers.colmap_file_handler import ColmapFileHandler
from photogrammetry_importer.importers.point_utility import add_points_as_mesh_vertices
```

Example 1: Add points contained in a `ply` file as a `particle system`.

```
import bpy
from photogrammetry_importer.file_handlers.point_data_file_handler import␣
↪PointDataFileHandler
from photogrammetry_importer.blender_utility.object_utility import add_collection
from photogrammetry_importer.importers.point_utility import add_points_as_mesh_vertices

ifp = "path/to/Blender-Addon-Photogrammetry-Importer/examples/Example.ply"
points = PointDataFileHandler.parse_point_data_file(ifp)
reconstruction_collection = add_collection("Reconstruction Collection")
add_points_as_mesh_vertices(
        points,
        reconstruction_collection=reconstruction_collection,
        add_mesh_to_point_geometry_nodes=True,
        point_radius=0.05,
        point_subdivisions=1,
        add_color_as_custom_property=True,
)
# Optionally, change the shading type to show the particle colors
area = next(area for area in bpy.context.screen.areas if area.type == "VIEW_3D")
space = next(space for space in area.spaces if space.type == "VIEW_3D")
space.shading.type = "RENDERED"
```

Example 2: Use the intrinsic and extrinsic parameters of each reconstructed camera to render the corresponding point cloud via an `off screen buffer` to disk.

```
import os
from photogrammetry_importer.file_handlers.colmap_file_handler import ColmapFileHandler
from photogrammetry_importer.types.point import Point
from photogrammetry_importer.blender_utility.object_utility import add_collection
from photogrammetry_importer.importers.camera_utility import add_camera_object
from photogrammetry_importer.opengl.utility import render_opengl_image
from photogrammetry_importer.blender_utility.image_utility import save_image_to_disk

# Make sure you've downloaded the corresponding images (i.e. the sceaux castle dataset)
model_idp = "path/to/Blender-Addon-Photogrammetry-Importer/examples/colmap_example_model_
↪bin"
image_idp = "path/to/Blender-Addon-Photogrammetry-Importer/examples/images"
odp = "path/to/output"

# Parse the reconstruction
cameras, points3D = ColmapFileHandler.parse_colmap_model_folder(model_idp, image_idp,␣
↪image_fp_type="NAME")
coords, colors = Point.split_points(points3D)

# Render the point cloud for reach camera
camera_collection = add_collection("Camera Collection")
render_img_name = "render_result"
for cam in cameras:
        cam_name = cam.get_file_name()
        print(f"Camera: {cam_name}")
        ofp = os.path.join(odp, cam_name)

        camera_object = add_camera_object(cam, cam_name, camera_collection)
```

(continues on next page)

(continued from previous page)

```
        render_opengl_image(render_img_name, camera_object, coords, colors, point_
↪size=10)
        save_image_to_disk(render_img_name, ofp, save_alpha=False)
```

Example 3: Use the animated camera to render the point cloud with `Cycles`.

```
import os
import bpy
from photogrammetry_importer.file_handlers.colmap_file_handler import ColmapFileHandler
from photogrammetry_importer.blender_utility.object_utility import add_collection
from photogrammetry_importer.importers.point_utility import add_points_as_object_with_
↪particle_system
from photogrammetry_importer.importers.camera_animation_utility import add_camera_
↪animation
from photogrammetry_importer.importers.camera_utility import adjust_render_settings_if_
↪possible

# Make sure you've downloaded the corresponding images (i.e. the sceaux castle dataset)
model_idp = "path/to/Blender-Addon-Photogrammetry-Importer/examples/colmap_example_model_
↪bin"
image_idp = "path/to/Blender-Addon-Photogrammetry-Importer/examples/images"
odp = "path/to/output"

# Parse the reconstruction
cameras, points3D = ColmapFileHandler.parse_colmap_model_folder(model_idp, image_idp,␣
↪image_fp_type="NAME")

# Add the reconstruction results
reconstruction_collection = add_collection("Reconstruction Collection")
add_points_as_object_with_particle_system(points3D, reconstruction_collection, point_
↪extent=0.02)
animated_camera_object = add_camera_animation(cameras, reconstruction_collection)

# Adjust the render settings and render animation with Cycles
adjust_render_settings_if_possible(cameras)
bpy.context.scene.render.engine = "CYCLES"
bpy.context.scene.cycles.device = "GPU"
bpy.context.scene.render.filepath = os.path.join(odp, "")
bpy.context.scene.camera = animated_camera_object
bpy.ops.render.render(animation=True)
```

1.10.2 Option 2: Call the appropriate operator registered in bpy.ops.import_scene

In Blender open the `Python Console` and use `Tabulator` to list the available operators with corresponding parameters, i.e.

```
>>> bpy.ops.import_scene.<TABULATOR>
>>> bpy.ops.import_scene.
                        colmap_model(
                        fbx(
                        gltf(
```

(continues on next page)

(continued from previous page)

```
                        meshroom_sfm_json(
                        mve_folder(
                        obj(
                        open3d_log_json(
                        openmvg_json(
                        opensfm_json(
                        point_data(
                        visualsfm_nvm(
                        x3d(
```

Or use `Tabulator` with a specific function, e.g. `point_data()`, to show the corresponding parameters.

```
>>> bpy.ops.import_scene.point_data(<TABULATOR>
>>> bpy.ops.import_scene.point_data(
point_data()
bpy.ops.import_scene.point_data(
        import_points=True,
        point_cloud_display_sparsity=1,
        draw_points_with_gpu=True,
        add_points_to_point_cloud_handle=True,
        add_points_as_particle_system=False,
        mesh_type='CUBE',
        point_extent=0.01,
        add_particle_color_emission=True,
        set_particle_color_flag=False,
        particle_overwrite_color=(0, 1, 0),
        add_points_as_mesh_oject=False,
        adjust_clipping_distance=False,
        filepath="",
        directory="",
        filter_glob="*.ply;*.pcd;*.las;*.laz;*.asc;*.pts;*.csv")
```

1.10.3 Python Scripting with Blender

VS Code with this extension has many advantages over Blender's built-in text editor. Here is an introduction / tutorial video.

Note: When using *VS Code* to start Blender with a specific addon for the first time, it is sometimes necessary to run the command twice (i.e. within VS Code run *ctrl+shift+p / Blender: Start / path_to_Blender_executable*, then close Blender, then run *ctrl+shift+p / Blender: Start / path_to_Blender_executable* again).

1.11 Extending the Addon

The addon follows a modular approach to simplify the extensibility of additional libraries. Each supported library requires the implementation of a corresponding *FileHandler* and *ImportOperator* - see the figure below. The *FileHandler* parses library specific file formats or directory structures and returns library agnostic information of cameras, points and meshes. The *ImportOperator* may use different classes provided by the framework (e.g. *CameraImporter*, *PointImporter* and *MeshImporter*) to define the required import options and to import the reconstruction extracted by the *FileHandler*.

Fig. 1: Integration of the addon with Blender - illustrated with the Colmap importer. In order to support additional SfM and MVS libraries it is sufficient to implement the corresponding import operators and file handlers. To simplify the figure only relevant classes and methods are shown.

1.12 Contribution

I am always happy to get advices as well as feature and pull requests. If you want to create a pull request, I recommend to use VS Code with this extension. This allows to perform fast development and validation cycles with Blender scripts and addons. Here is an introduction / tutorial video.

This addon relies on Black for formatting. To ensure that your `Pull Request` is correctly formatted perform the following steps:

- Install Black - checkout the installation instructions
- `cd path/to/Blender-Addon-Photogrammetry-Importer`
- `black --line-length 79 --exclude photogrammetry_importer/ext photogrammetry_importer`
- `black --line-length 79 doc/sphinx/source/conf.py`

The addon uses Docstrings for documentation - see PEP 257 and PEP 287 .

1.13 Documentation

The addon uses Sphinx, sphinx_rtd_theme, sphinx-autoapi to generate the documentation.

These tools can be installed with

- `pip install -U sphinx`
- `pip install sphinx-rtd-theme`
- `pip install sphinx-autoapi`

In order to build the documentation locally

- download the project
- navigate to the `sphinx` folder (i.e. `cd Blender-Addon-Photogrammetry-Importer/doc/sphinx`)
- run `make html` / `make latex` (using Linux) or `make.bat html` / `make.bat latex` (using Windows)

1.14 Recent Features / Changelog

Changelog with most relevant features. Recently added features are listed at the top of this page.

1.14.1 2022

- Fix geometry node creation for Blender 3.2.1
- Add support for installation and removal of individual dependencies
- Fix the point exporter for OpenGL point clouds
- Remove the particle system based point cloud representation method (because of the new point clound representation option based on geometry nodes)
- Add particle colors to geometry nodes (Thanks to @Linusnie)
- Add a new point cloud representation method based on geometry nodes. Allows to render the point coud with Eevee and Cycles
- Replace pylas with laspy (>= 2.0)
- Add an option to center points around the origin (useful for laz/las files)
- Remove the point color computation for OpenMVG json files, due to a bug in pillow caused by the blender python environment
- Adjust the screenshot rendering functionality according to api changes of Blender 3.0
- Replacing the bgl module with the gpu module to fix the offscreen rendering of point clouds under windows
- Fix the dependency installation for Blender 2.8 - 2.91 of the newly introduced dependency manager

1.14.2 2021

- Improve management of dependency installation and corresponding GUI options
- Made point sizes of point clouds and depth maps (drawn with OpenGL) persistent
- Changed the usage of draw handlers to avoid potential crashes when deleting the point cloud anchor objects
- Made depth maps persistent (i.e. the corresponding information is stored in the blend file)
- Added features to export images of the imported reconstructions including cameras, background images, image planes, point clouds and meshes

1.14.3 2020

- Reorganized (persistent) addon preferences
- Added an option to use the undistorted images contained in the workspaces of the Colmap, Meshroom and MVE
- Added several python examples that demonstrate the API usage
- Added vertex colors to the mesh shader nodes to improve the visibility of the corresponding mesh
- Added background images for the animated camera
- Added code to automatically generate the API Documentation with autoapi
- Fixed an incorrect offset in the texture coordinate computation of the particle system
- Added a workaround to circumvent a bug in Blender, which appears only for large particle systems (T81103)
- Added GUI elements to install/uninstall the dependencies (Pillow, Pyntcloud)
- Addon uses now the Pyntcloud library to import PLY, PCD, LAS, ASC, PTS and CSV files
- Added an option to import depth maps of MVE workspaces

- Added an option to import depth maps of Colmap as point clouds
- Added support for MVE workspaces
- Added addon preferences to configure the import/export default settings
- Added addon preferences to enable/disable importers and exporters
- Added an OpenSfM importer
- OpenGL data is now persistent (stored in blend file) and is available after reopening
- Added a panel with options to export renderings of the point cloud using OpenGL
- Added support for Colmap dense workspaces
- Added support for Meshroom projects (.mg files)
- Fixed occlusion of point clouds drawn with OpenGL
- Added a Colmap exporter
- Added an Open3D importer
- Added an option to render point clouds with OpenGL
- Added support for absolute and relative paths in reconstruction results
- Added a preset possiblity for each importer to customize default import options

1.14.4 2019

- Added support to import undistorted images of Colmap/Meshroom
- Fixed a bug leading to incorrect principal points
- Added option to remove discontinuities in animations
- Added an option to show source images as Blender background images
- Added particle emission to improve visibility
- Added importers for Colmap, OpenMVG and Meshroom
- Compatibility fix for Blender 2.8

1.14.5 2018

- Added an option to add camera animation
- Added an option to import images as image planes
- Added an exporter for cameras and mesh vertex positions as NVM
- Added an option to represent point clouds with particle systems

1.14.6 2017

- Added a NVM importer
- Initial Commit

1.15 API Reference

This page contains auto-generated API reference documentation[1].

1.15.1 photogrammetry_importer

A Blender addon to import different photogrammetry formats.

Subpackage Summary

ext
External dependencies.

file_handlers
Classes to read and write different file formats.

operators
Operators to import and export different file formats into Blender.

panels
GUI elements to adjust and leverage the imported objects.

preferences
Persistent addon preferences.

properties
Properties used by the import and export operators.

registration
Registration of the import and export operators.

types
Types used by different subpackages.

utility
General and Blender-specific utility functions.

Subpackages

photogrammetry_importer.blender_utility

Contains Blender-specific utility functions.

[1] Created with sphinx-autoapi

Submodules

photogrammetry_importer.blender_utility.image_utility

Module Contents

photogrammetry_importer.blender_utility.image_utility.**save_image_to_disk**(*image_name*, *file_path*, *save_alpha=True*)

Save a Blender image to disk.

photogrammetry_importer.blender_utility.logging_utility

Module Contents

photogrammetry_importer.blender_utility.logging_utility.**log_report**(*output_type*, *some_str*, *op=None*)

Write a string to the console and to Blender's info area.

photogrammetry_importer.blender_utility.object_utility

Module Contents

photogrammetry_importer.blender_utility.object_utility.**add_empty**(*empty_name*, *collection=None*)

Add an empty to the scene.

photogrammetry_importer.blender_utility.object_utility.**add_obj**(*data*, *obj_name*, *collection=None*)

Add an object to the scene.

photogrammetry_importer.blender_utility.object_utility.**add_collection**(*collection_name*, *parent_collection=None*)

Add a collection to the scene.

photogrammetry_importer.blender_utility.retrieval_utility

Module Contents

photogrammetry_importer.blender_utility.retrieval_utility.**get_selected_object**()

Get the selected object or return None.

photogrammetry_importer.blender_utility.retrieval_utility.**get_selected_empty**()

Get the selected empty or return None.

photogrammetry_importer.blender_utility.retrieval_utility.**get_selected_camera**()

Get the selected camera or return None.

photogrammetry_importer.blender_utility.retrieval_utility.**get_scene_animation_indices**()

Get the animation indices of the scene.

photogrammetry_importer.blender_utility.retrieval_utility.**get_object_animation_indices**(*obj*)

Get the animation indices of the object.

photogrammetry_importer.file_handlers

Contains classes to read and write different file formats.

Submodules

photogrammetry_importer.file_handlers.colmap_file_handler

Module Contents

class photogrammetry_importer.file_handlers.colmap_file_handler.**ColmapFileHandler**

Class to read and write Colmap models and workspaces.

static parse_colmap_model_folder(*model_idp*, *image_dp*, *image_fp_type*, *depth_map_idp=None*, *suppress_distortion_warnings=False*, *op=None*)

Parse a Colmap model.

static parse_colmap_folder(*idp*, *use_workspace_images*, *image_dp*, *image_fp_type*, *suppress_distortion_warnings=False*, *op=None*)

Parse a Colmap model or a Colmap workspace.

static write_colmap_model(*odp*, *cameras*, *points*, *op=None*)

Write cameras and points as Colmap model.

photogrammetry_importer.file_handlers.image_file_handler

Module Contents

class photogrammetry_importer.file_handlers.image_file_handler.**ImageFileHandler**

Class to read and write images using Pillow.

PILImage

classmethod read_image_size(*image_ifp*, *default_width*, *default_height*, *op=None*)

Read image size from disk.

photogrammetry_importer.file_handlers.meshroom_file_handler

Module Contents

class photogrammetry_importer.file_handlers.meshroom_file_handler.**MeshroomFileHandler**

Class to read and write Meshroom files and workspaces.

classmethod parse_meshroom_sfm_file(*sfm_ifp*, *image_idp*, *image_fp_type*, *suppress_distortion_warnings*, *op=None*)

Parse a `Meshroom` (`.sfm` or `.json`) file.

Parse different file formats created with the `StructureFromMotion` / `ConvertSfMFormat` node in `Meshroom`.

classmethod parse_meshrom_mg_file(*mg_fp*, *sfm_node_type*, *sfm_node_number*, *mesh_node_type*, *mesh_node_number*, *prepare_node_number*, *op=None*)

Parse a `Meshroom` project file (`.mg`).

classmethod parse_meshroom_file(*meshroom_ifp*, *use_workspace_images*, *image_dp*, *image_fp_type*, *suppress_distortion_warnings*, *sfm_node_type*, *sfm_node_number*, *mesh_node_type*, *mesh_node_number*, *prepare_node_number*, *op=None*)

Parse a `Meshroom` file.

Supported file formats are `.mg`, `.sfm` or `.json`.

photogrammetry_importer.file_handlers.mve_file_handler

Module Contents

class photogrammetry_importer.file_handlers.mve_file_handler.**MVEFileHandler**

Class to read and write `MVE` workspaces.

static parse_synth_out(*synth_out_ifp*)

Parse the `synth_0.out` file in the `MVE` workspace.

static parse_meta(*meta_ifp*, *width*, *height*, *camera_name*, *op*)

Parse a `meta.ini` file in the `MVE` workspace.

static parse_views(*views_idp*, *default_width*, *default_height*, *add_depth_maps_as_point_cloud*, *op=None*)

Parse the `views` directory in the `MVE` workspace.

static parse_mve_workspace(*workspace_idp*, *default_width*, *default_height*, *add_depth_maps_as_point_cloud*, *suppress_distortion_warnings*, *op=None*)

Parse a `MVE` workspace.

static read_depth_map(*depth_map_ifp*)

Read a depth map.

photogrammetry_importer.file_handlers.open3D_file_handler

Module Contents

class photogrammetry_importer.file_handlers.open3D_file_handler.**Open3DFileHandler**

Class to read and write `Open3D` files.

static parse_open3d_file(*open3d_ifp*, *image_dp*, *image_fp_type*, *op*)

Parse an Open3D (.json or .log) file.

The .json format supports intrinsics as well as extrinsic parameters, whereas the .log (Redwood) format contains only extrinsic parameters.

photogrammetry_importer.file_handlers.openmvg_json_file_handler

Module Contents

class photogrammetry_importer.file_handlers.openmvg_json_file_handler.**OpenMVGJSONFileHandler**

Class to read and write OpenMVG files.

static parse_openmvg_file(*input_openMVG_file_path*, *image_dp*, *image_fp_type*, *suppress_distortion_warnings*, *op=None*)

Parse an OpenMVG (.json) file.

photogrammetry_importer.file_handlers.opensfm_json_file_handler

Module Contents

class photogrammetry_importer.file_handlers.opensfm_json_file_handler.**OpenSfMJSONFileHandler**

Class to read and write OpenSfM files.

static parse_opensfm_file(*input_opensfm_fp*, *image_dp*, *image_fp_type*, *reconstruction_idx*, *suppress_distortion_warnings=False*, *op=None*)

Parse a OpenSfM (.json) file.

photogrammetry_importer.file_handlers.point_data_file_handler

Module Contents

class photogrammetry_importer.file_handlers.point_data_file_handler.**PointDataFileHandler**

Class to read and write common point data files.

static parse_point_data_file(*ifp*, *op=None*)

Parse a point data file.

Supported file formats are: .ply, .pcd, .las, .laz, .asc, .pts and .csv.

Relies on the pyntcloud, the laspy and/or the lazrs library to parse the different file formats.

photogrammetry_importer.file_handlers.transformation_file_handler

Module Contents

class photogrammetry_importer.file_handlers.transformation_file_handler.**TransformationFileHandler**

Class to read directories with files storing transformations.

static parse_transformation_folder(*t_idp*, *op=None*)

Parse a directory with files storing transformations.

photogrammetry_importer.file_handlers.utility

Module Contents

photogrammetry_importer.file_handlers.utility.**check_radial_distortion**(*radial_distortion*, *camera_name*, *op=None*)

Check if the radial distortion is compatible with Blender.

photogrammetry_importer.file_handlers.visualsfm_file_handler

Module Contents

class photogrammetry_importer.file_handlers.visualsfm_file_handler.**VisualSfMFileHandler**

Class to read and write `VisualSfM` files.

classmethod parse_visualsfm_file(*input_visual_fsm_file_name*, *image_dp*, *image_fp_type*, *suppress_distortion_warnings*, *op=None*)

Parse a `VisualSfM` (`.nvm`) file.

classmethod write_visualsfm_file(*output_nvm_file_name*, *cameras*, *points*, *op=None*)

Write cameras and points as `.nvm` file.

photogrammetry_importer.importers

Contains classes useful for import operators.

Submodules

photogrammetry_importer.importers.camera_animation_utility

Module Contents

photogrammetry_importer.importers.camera_animation_utility.**add_camera_animation**(*cameras*, *parent_collection*, *animation_frame_source='ORIGINA*, *add_background_images=Fals*, *reorganize_undistorted_images=Fals*, *number_interpolation_frames=0*, *interpolation_type='LINEAR'*, *remove_rotation_discontinuities=*, *consider_missing_cameras_during*, *image_dp=None*, *image_fp_type=None*, *op=None*)

Add an animated camera from a set of reconstructed cameras.

photogrammetry_importer.importers.camera_importer

Module Contents

class photogrammetry_importer.importers.camera_importer.**CameraImporter**

Importer for cameras and corresponding image information.

use_workspace_images :BoolProperty(name='Use Workspace Images', description='If selected, use the (undistorted) images in the workspace (if available). Otherwise use the images in the default image path.', default=True)

image_fp_items = [None, None, None]

image_fp_type :EnumProperty(name='Image File Path Type', description='Choose how image file paths are treated, i.e. absolute path, relative path or file name', items=image_fp_items)

image_dp :StringProperty(name='Image Directory', description='Assuming that the SfM reconstruction result is located in <some/path/rec.ext> or <some/path/rec_directory>. The addons uses either <some/path/images> (if available) or <some/path> as default image path. For MVS reconstruction results of Colmap, Meshroom or MVE the addon may or may not search for the images inside the corresponding workspace', default='')

import_cameras :BoolProperty(name='Import Cameras', description='Import Cameras', default=True)

default_width :IntProperty(name='Default Width', description='Width, which will be used used if corresponding image is not found', default=-1)

```
default_height :IntProperty(name='Default Height', description='Height, which will
be used used if corresponding image is not found', default=-1)

default_focal_length :FloatProperty(name='Focal length in pixel', description='Value
for missing focal length in LOG (Open3D) file. ', default=float('nan'))

default_pp_x :FloatProperty(name='Principal Point X Component',
description='Principal Point X Component, which will be used if not contained in the
NVM (VisualSfM) / LOG (Open3D) file. If no value is provided, the principal point is
set to the image center', default=float('nan'))

default_pp_y :FloatProperty(name='Principal Point Y Component',
description='Principal Point Y Component, which will be used if not contained in the
NVM (VisualSfM) / LOG (Open3D) file. If no value is provided, the principal point is
set to the image center', default=float('nan'))

add_background_images :BoolProperty(name='Add a Background Image for each Camera',
description='The background image is only visible by viewing the scene from a
specific camera', default=True)

add_image_planes :BoolProperty(name='Add an Image Plane for each Camera',
description='Add an Image Plane for each Camera - only for non-panoramic cameras',
default=False)

add_image_plane_emission :BoolProperty(name='Add Image Plane Color Emission',
description='Add image plane color emission to increase the visibility of the image
planes', default=True)

image_plane_transparency :FloatProperty(name='Image Plane Transparency Value',
description='Transparency value of the image planes:  0 = invisible, 1 = opaque',
default=0.5, min=0, max=1)

add_depth_maps_as_point_cloud :BoolProperty(name='Add Depth Maps (EXPERIMENTAL)',
description='Add the depth map (if available) as point cloud for each Camera',
default=False)

use_default_depth_map_color :BoolProperty(name='Use Default Depth Map Color',
description='If not selected, each depth map is colorized with a different (random)
color', default=False)

depth_map_default_color :FloatVectorProperty(name='Depth Map Color',
description='Depth map color', subtype='COLOR', size=3, default=(0.0, 1.0, 0.0),
min=0.0, max=1.0)

depth_map_display_sparsity :IntProperty(name='Depth Map Display Sparsity',
description='Adjust the sparsity of the depth maps. A value of 10 means that every
10th depth map value is converted to a 3D point', default=10, min=1)

depth_map_id_or_name_str :StringProperty(name='Depth Map IDs or Names to Display',
description='A list of camera indices or names (separated by whitespaces) used to
select the depth maps, which will be displayed as point clouds. If no indices are
provided, all depth maps are shown. The names must not contain whitespaces',
default='')
```

add_camera_motion_as_animation :BoolProperty(name='Add Camera Motion as Animation', description='Add an animation reflecting the camera motion. The order of the cameras is determined by the corresponding file name', default=True)

animation_frame_source :EnumProperty(name='Use original frames', items=(('ORIGINAL', 'Original Frames', ''), ('ADJUSTED', 'Adjusted Frames', '')))

add_animated_camera_background_images :BoolProperty(name='Add Background Images for the Animated Camera', description='The background images are only visible by viewing the scene from the animated camera at the corresponding time step', default=True)

reorganize_undistorted_images :BoolProperty(name='Reorganize Undistorted Workspace Images', description='Rename the undistorted images according to the original image names and write them to a single directory. Certain libraries such as Meshroom or MVE rename or move the undistorted images to different directories. Thus, the reversal is necessary to use the images as background sequence for the animated camera. WARNING: This will write a copy of the corresponding images to the workspace directory', default=True)

number_interpolation_frames :IntProperty(name='Number of Frames Between two Reconstructed Cameras', description='The poses of the animated camera are interpolated', default=0, min=0)

interpolation_items = [['LINEAR', 'LINEAR', '', 1], ['BEZIER', 'BEZIER', '', 2], ['SINE', 'SINE', '', 3], ['QUAD',...

interpolation_type :EnumProperty(name='Interpolation Type', description='Blender string that defines the type of the interpolation', items=interpolation_items)

consider_missing_cameras_during_animation :BoolProperty(name='Adjust Frame Numbers of Camera Animation', description='Assume there are three consecutive images A,B and C, but only A and C have been reconstructed. This option adjusts the frame number of C and the number of interpolation frames between camera A and C', default=True)

remove_rotation_discontinuities :BoolProperty(name='Remove Rotation Discontinuities', description='The addon uses quaternions q to represent the rotation. A quaternion q and its negative -q describe the same rotation. This option allows to remove different signs', default=True)

suppress_distortion_warnings :BoolProperty(name='Suppress Distortion Warnings', description='Radial distortion might lead to incorrect alignments of cameras and points. Enable this option to suppress corresponding warnings. If possible, consider to re-compute the reconstruction using a camera model without radial distortion', default=False)

adjust_render_settings :BoolProperty(name='Adjust Render Settings', description='Adjust the render settings according to the corresponding images - all images have to be captured with the same device. If disabled the visualization of the camera cone in 3D view might be incorrect', default=True)

camera_extent :FloatProperty(name='Initial Camera Extent (in Blender Units)', description='Initial Camera Extent (Visualization)', default=1)

draw_camera_options(*layout*, *draw_workspace_image_usage=False*, *reorganize_undistorted_images=False*, *draw_image_fp=True*, *draw_depth_map_import=False*, *draw_image_size=False*, *draw_principal_point=False*, *draw_focal_length=False*, *draw_everything=False*)

Draw camera import options.

set_intrinsics_of_cameras(*cameras*)

Set intrinsic parameters of cameras.

This function should be overwritten, if the intrinsic parameters are not part of the reconstruction data (e.g. log file).

set_image_size_of_cameras(*cameras*)

Set image size of cameras.

This function should be overwritten, if the image size is not part of the reconstruction data (e.g. nvm file).

import_photogrammetry_cameras(*cameras*, *parent_collection*)

Import the cameras using the properties of this class.

photogrammetry_importer.importers.camera_utility

Module Contents

photogrammetry_importer.importers.camera_utility.**compute_principal_point_shift**(*camera*, *relativ_to_largest_extend*)

Return the shift of the principal point in the 3D view port.

photogrammetry_importer.importers.camera_utility.**adjust_render_settings_if_possible**(*cameras*, *op=None*)

Adjust the render settings according to the camera parameters.

photogrammetry_importer.importers.camera_utility.**add_camera_object**(*camera*, *camera_name*, *camera_collection*, *copy_matrix_world=True*)

Add a camera as Blender object.

photogrammetry_importer.importers.camera_utility.**invert_y_and_z_axis**(*input_matrix_or_vector*)

Invert the y and z axis of a given matrix or vector.

Many SfM / MVS libraries use coordinate systems that differ from Blender's coordinate system in the y and the z coordinate. This function inverts the y and the z coordinates in the corresponding matrix / vector entries, which is equivalent to a rotation by 180 degree around the x axis.

photogrammetry_importer.importers.camera_utility.**compute_camera_matrix_world**(*camera*, *convert_coordinate_system=True*)

Compute Blender's `matrix_world` for a given camera.

photogrammetry_importer.importers.camera_utility.**add_cameras**(*cameras*, *parent_collection*, *add_background_images=False*, *add_image_planes=False*, *add_depth_maps_as_point_cloud=True*, *convert_camera_coordinate_system=True*, *camera_collection_name='Cameras'*, *image_plane_collection_name='Image Planes'*, *depth_map_collection_name='Depth Maps'*, *camera_scale=1.0*, *image_plane_transparency=0.5*, *add_image_plane_emission=True*, *depth_map_point_size=1*, *use_default_depth_map_color=False*, *depth_map_default_color=(1.0, 0.0, 0.0)*, *depth_map_display_sparsity=10*, *depth_map_id_or_name_str=''*, *op=None*)

Add a set of reconstructed cameras to Blender's 3D view port.

photogrammetry_importer.importers.camera_utility.**add_camera_image_plane**(*matrix_world*, *blender_image*, *camera*, *name*, *transparency*, *add_image_plane_emission*, *image_planes_collection*, *op=None*)

Add an image plane corresponding to a reconstructed camera.

photogrammetry_importer.importers.mesh_importer

Module Contents

class photogrammetry_importer.importers.mesh_importer.**MeshImporter**

Importer for meshes.

import_mesh :BoolProperty(name='Import Mesh', description='Import mesh (if available). Only relevant for files/folders referencing/containing mesh files (such as *.mg files of Meshroom or dense Colmap folders). Note that Blenders build-in ply- and obj-importer are quite slow', default=False)

add_mesh_color_emission :BoolProperty(name='Add Color Emission of Mesh', description='Enabling color emission improves the visibility of the mesh colors', default=True)

draw_mesh_options(*layout*)

Draw mesh import options.

import_photogrammetry_mesh(*mesh_fp*, *reconstruction_collection*)

Import a mesh using the properties of this class.

photogrammetry_importer.importers.mesh_utility

Module Contents

photogrammetry_importer.importers.mesh_utility.**add_color_emission_to_material**(*mesh_obj*)

Add color emmision for the given mesh to improve the visibility.

photogrammetry_importer.importers.mesh_utility.**add_mesh_vertex_color_material**(*mesh_obj*, *mesh_material_name*, *add_mesh_color_emission*)

Add a material with vertex colors to the given mesh.

photogrammetry_importer.importers.point_importer

Module Contents

class photogrammetry_importer.importers.point_importer.**PointImporter**

Importer for points.

import_points :BoolProperty(name='Import Points', description='Import Points', default=True)

point_cloud_display_sparsity :IntProperty(name='Point Cloud Display Sparsity', description='Adjust the sparsity of the point cloud. A value of n means that every n-th point in the point cloud is added', default=1, min=1)

center_points :BoolProperty(name='Center Data Around Origin', description='Center data by subtracting the centroid. Useful for las/laz files, which contain usually large offsets.', default=False)

draw_points_with_gpu :BoolProperty(name='Draw Points in the 3D View with OpenGL.', description='Draw Points in the 3D View. Allows to visualize point clouds with many elements. These are not visible in eevee/cycles renderings.', default=True)

add_points_to_point_cloud_handle :BoolProperty(name='Add point data to the point cloud handle.', description='This allows to draw the point cloud (again) with OpenGL after saving and reloading the blend file.', default=True)

point_size :IntProperty(name='Initial Point Size', description='Initial Point Size', default=5)

add_points_as_mesh_oject :BoolProperty(name='Add Points as Mesh Object', description='Use a mesh object to represent the point cloud with the vertex positions.', default=False)

add_mesh_to_point_geometry_nodes :BoolProperty(name='Add Geometry Nodes', description="Add Geometry Nodes to allow rendering of the point cloud with Blender's built-in renderers (Eevee / Cycles).", default=True)

point_radius :FloatProperty(name='Initial Point Radius', description='Initial point radius (can be changed in GUI).', default=0.05)

point_subdivisions :IntProperty(name='Initial Point Subdivisions', description='Initial point subdivisions (can be changed in GUI).', default=1)

add_color_as_custom_property :BoolProperty(name='Add Colors as Custom Property', description='Use a custom property (named colors) to store the point cloud colors.', default=True)

draw_point_options(*layout*, *draw_everything=False*)

Draw point import options.

import_photogrammetry_points(*points*, *reconstruction_collection*)

Import a point cloud using the properties of this class.

photogrammetry_importer.importers.point_utility

Module Contents

photogrammetry_importer.importers.point_utility.**add_points_as_object_with_particle_system**(*points*, *reconstruction_collection*, *mesh_type='CUB*, *point_extent=0.0*, *add_particle_col*, *particle_overwrite_co*, *op=None*)

Add a point cloud as particle system.

This method is deprecated. It is recommended to use `add_points_as_mesh_vertices()` instead.

photogrammetry_importer.importers.point_utility.**create_geometry_nodes_node_group**()

Create a new node group for a geometry nodes modifier.

photogrammetry_importer.importers.point_utility.**add_points_as_mesh_vertices**(*points*, *reconstruction_collection*, *add_mesh_to_point_geometry_node*, *point_radius=0.05*, *point_subdivisions=1*, *add_color_as_custom_property=Tr*, *op=None*)

Add a point cloud as mesh.

photogrammetry_importer.opengl

Contains OpenGL functions to render point clouds.

Submodules

photogrammetry_importer.opengl.draw_manager

Module Contents

class photogrammetry_importer.opengl.draw_manager.**DrawManager**

Class that allows to represent point clouds with OpenGL in Blender.

classmethod get_singleton()

Return a singleton of this class.

register_points_draw_callback(*object_anchor*, *coords*, *colors*, *point_size*)

Register a callback to draw a point cloud.

get_coords_and_colors(*visible_only=False*)

Return the coordinates and the colors of the maintained points.

delete_anchor(*object_anchor*)

Delete the anchor used to control the pose of the point cloud.

get_draw_callback_handler(*object_anchor*)

Get the draw callback handler corresponding to the object anchor.

photogrammetry_importer.opengl.utility

Module Contents

photogrammetry_importer.opengl.utility.**draw_points**(*points*, *point_size*, *add_points_to_point_cloud_handle*, *reconstruction_collection=None*, *object_anchor_handle_name='OpenGL Point Cloud'*, *op=None*)

Draw points using OpenGL.

photogrammetry_importer.opengl.utility.**draw_coords**(*coords*, *color=(0, 0, 255, 1.0)*, *point_size=1*, *add_points_to_point_cloud_handle=True*, *reconstruction_collection=None*, *object_anchor_handle_name='OpenGL Coord Point Cloud'*, *op=None*)

Draw coordinates using OpenGL.

photogrammetry_importer.opengl.utility.**redraw_points**(*dummy*)

Redraw points of the previous Blender session.

photogrammetry_importer.opengl.utility.**render_opengl_image**(*image_name*, *cam*, *coords*, *colors*, *point_size*)

Render the given coordinates with OpenGL.

photogrammetry_importer.operators

Contains Operators to import and export different formats into Blender.

Submodules

photogrammetry_importer.operators.colmap_export_op

Module Contents

class photogrammetry_importer.operators.colmap_export_op.**ExportColmapOperator**

Bases: *photogrammetry_importer.operators.export_op.ExportOperator*, bpy_extras.io_utils.ExportHelper

Blender operator to export a Colmap model.

bl_idname = export_scene.colmap

bl_label = Export Colmap

bl_options

directory :StringProperty()

files :CollectionProperty(name='Directory Path', description='Directory path used for exporting the Colmap model', type=bpy.types.OperatorFileListElement)

filename_ext =

execute(*context*)

Export selected cameras and points as Colmap model.

photogrammetry_importer.operators.colmap_import_op

Module Contents

class photogrammetry_importer.operators.colmap_import_op.**ImportColmapOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.importers.mesh_importer.MeshImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*

Blender operator to import a Colmap model/workspace.

bl_idname = import_scene.colmap_model

bl_label = Import Colmap Model Folder

bl_options

directory :StringProperty()

execute(*context*)

Import a Colmap model/workspace.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.export_op

Module Contents

class photogrammetry_importer.operators.export_op.**ExportOperator**

Bases: bpy.types.Operator

Abstract basic export operator.

get_selected_cameras_and_vertices_of_meshes(*odp*)

Get selected cameras and vertices.

abstract execute(*context*)

Abstract method that must be overriden by a subclass.

photogrammetry_importer.operators.general_options

Module Contents

class photogrammetry_importer.operators.general_options.**GeneralOptions**

Class to define and apply general options.

adjust_clipping_distance :BoolProperty(name='Adjust Clipping Distance', description='Adjust clipping distance of 3D view.', default=False)

draw_general_options(*layout*)

Draw general options.

apply_general_options()

Apply the options defined by this class.

photogrammetry_importer.operators.import_op

Module Contents

class photogrammetry_importer.operators.import_op.**ImportOperator**

Bases: bpy.types.Operator

Abstract basic import operator.

initialize_options_from_addon_preferences()

Initialize the import options from the current addon preferences.

get_default_image_path(*reconstruction_fp*, *image_dp*)

Get the (default) path that defines where to look for images.

abstract execute(*context*)

Abstract method that must be overriden by a subclass.

photogrammetry_importer.operators.meshroom_import_op

Module Contents

class photogrammetry_importer.operators.meshroom_import_op.**ImportMeshroomOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.importers.mesh_importer.MeshImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*, bpy_extras.io_utils.ImportHelper

Import a Meshroom MG/SfM/JSON file.

bl_idname = import_scene.meshroom_sfm_json

bl_label = Import Meshroom SfM/JSON/MG

bl_options

filepath :StringProperty(name='Meshroom JSON File Path', description='File path used for importing the Meshroom SfM/JSON/MG' + ' file')

directory :StringProperty()

filter_glob :StringProperty(default='*.sfm;*.json;*.mg', options={'HIDDEN'})

sfm_node_items = [['AUTOMATIC', 'AUTOMATIC', '', 1], ['ConvertSfMFormatNode', 'ConvertSfMFormatNode', '', 2],...

sfm_node_type :EnumProperty(name='Structure From Motion Node Type', description='Use this property to select the node with the structure' + ' from motion results to import', items=sfm_node_items)

sfm_node_number :IntProperty(name='ConvertSfMFormat Node Number', description='Use this property to select the desired node.' + ' By default the node with the highest number is imported.', default=-1)

mesh_node_items = [['AUTOMATIC', 'AUTOMATIC', '', 1], ['Texturing', 'Texturing', '', 2], ['MeshFiltering',...

mesh_node_type :EnumProperty(name='Mesh Node Type', description='Use this property to select the node with the mesh' + ' results to import', items=mesh_node_items)

mesh_node_number :IntProperty(name='Mesh Node Number', description='Use this property to select the desired node.' + ' By default the node with the highest number is imported.', default=-1)

prepare_node_number :IntProperty(name='Prepare Dense Node Number', description='Use this property to select the desired node.' + ' By default the node with the highest number is imported.', default=-1)

execute(*context*)

Import a Meshroom file/workspace.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.mve_import_op

Module Contents

class photogrammetry_importer.operators.mve_import_op.**ImportMVEOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*

Import a Multi-View Environment reconstruction folder.

bl_idname = import_scene.mve_folder

bl_label = Import MVE Folder

bl_options

directory :StringProperty()

execute(*context*)

Import an MVE workspace.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.open3d_import_op

Module Contents

class photogrammetry_importer.operators.open3d_import_op.**ImportOpen3DOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*, bpy_extras.io_utils.ImportHelper

Import an Open3D LOG/JSON file

bl_idname = import_scene.open3d_log_json

bl_label = Import Open3D LOG/JSON

bl_options

filepath :StringProperty(name='Open3D LOG/JSON File Path', description='File path used for importing the Open3D LOG/JSON file')

directory :StringProperty()

filter_glob :StringProperty(default='*.log;*.json', options={'HIDDEN'})

set_intrinsics_of_cameras(*cameras*)

Enhances the imported cameras with intrinsic information.

Overwrites the method in `CameraImporter`.

set_image_size_of_cameras(*cameras*)

Enhance the imported cameras with image related information.

Overwrites the method in `CameraImporter`.

execute(*context*)

Import an `Open3D` file.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.openmvg_import_op

Module Contents

class photogrammetry_importer.operators.openmvg_import_op.**ImportOpenMVGOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*, `bpy_extras.io_utils.ImportHelper`

Import an `OpenMVG` JSON file

bl_idname = import_scene.openmvg_json

bl_label = Import OpenMVG JSON

bl_options

filepath :StringProperty(name='OpenMVG JSON File Path', description='File path used for importing the OpenMVG JSON file')

directory :StringProperty()

filter_glob :StringProperty(default='*.json', options={'HIDDEN'})

execute(*context*)

Import an OpenMVG JSON file.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.opensfm_import_op

Module Contents

class photogrammetry_importer.operators.opensfm_import_op.**ImportOpenSfMOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*, bpy_extras.io_utils.ImportHelper

Import an OpenSfM JSON file

bl_idname = import_scene.opensfm_json

bl_label = Import OpenSfM JSON

bl_options

filepath :StringProperty(name='OpenSfM JSON File Path', description='File path used for importing the OpenSfM JSON file')

directory :StringProperty()

filter_glob :StringProperty(default='*.json', options={'HIDDEN'})

reconstruction_number :IntProperty(name='Reconstruction Number', description='If the input file contains multiple reconstructions, use' + ' this property to select the desired reconstruction.', default=0)

execute(*context*)

Import an OpenSfM JSON file.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.point_data_import_op

Module Contents

class photogrammetry_importer.operators.point_data_import_op.**ImportPointDataOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*, bpy_extras.io_utils.ImportHelper

Import point data (e.g. a PLY file) as point cloud.

bl_idname = import_scene.point_data

bl_label = Import Point Data

bl_options

filepath :StringProperty(name='Point Data File Path', description='File path used for importing the point data file')

directory :StringProperty()

filter_glob :StringProperty(default='*.ply;*.pcd;*.las;*.laz;*.asc;*.pts;*.csv', options={'HIDDEN'})

execute(*context*)

Import a file with point data (e.g. PLY).

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.operators.utility

Module Contents

photogrammetry_importer.operators.utility.**set_image_size_for_cameras**(*cameras*, *default_width*, *default_height*, *op=None*)

Set image sizes for cameras and return a boolean.

photogrammetry_importer.operators.visualsfm_export_op

Module Contents

class photogrammetry_importer.operators.visualsfm_export_op.**ExportVisualSfMOperator**

Bases: *photogrammetry_importer.operators.export_op.ExportOperator*, bpy_extras.io_utils.ExportHelper

Export a VisualSfM file.

bl_idname = export_scene.nvm

bl_label = Export NVM

bl_options

directory :StringProperty()

files :CollectionProperty(name='File Path', description='File path used for exporting the NVM file', type=bpy.types.OperatorFileListElement)

filename_ext = .nvm

filter_glob :StringProperty(default='*.nvm', options={'HIDDEN'})

execute(*context*)

Export selected cameras and points as `VisualSfM` file.

photogrammetry_importer.operators.visualsfm_import_op

Module Contents

class photogrammetry_importer.operators.visualsfm_import_op.**ImportVisualSfMOperator**

Bases: *photogrammetry_importer.operators.import_op.ImportOperator*, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.operators.general_options.GeneralOptions*, bpy_extras.io_utils.ImportHelper

Import a `VisualSfM` NVM file.

bl_idname = import_scene.visualsfm_nvm

bl_label = Import NVM

bl_options

filepath :StringProperty(name='NVM File Path', description='File path used for importing the NVM file')

directory :StringProperty()

filter_glob :StringProperty(default='*.nvm', options={'HIDDEN'})

set_image_size_of_cameras(*cameras*)

Enhance the imported cameras with image related information.

Overwrites the method in `CameraImporter`.

execute(*context*)

Import an `VisualSfM` file.

invoke(*context*, *event*)

Set the default import options before running the operator.

draw(*context*)

Draw the import options corresponding to this operator.

photogrammetry_importer.panels

Contains GUI elements to adjust and leverage the imported objects.

Submodules

photogrammetry_importer.panels.render_operators

Module Contents

class photogrammetry_importer.panels.render_operators.**SaveOpenGLRenderImageOperator**

Bases: bpy.types.Operator

An Operator to save a rendering of the point cloud as Blender image.

bl_idname = photogrammetry_importer.save_opengl_render_image

bl_label = Save as Blender Image

bl_description = Use a single camera to render the point cloud.

classmethod poll(*context*)

Return the availability status of the operator.

execute(*context*)

Render the point cloud and save the result as image in Blender.

class photogrammetry_importer.panels.render_operators.**ExportOpenGLRenderImageOperator**

Bases: bpy.types.Operator, bpy_extras.io_utils.ExportHelper

An Operator to save a rendering of the point cloud to disk.

bl_idname = photogrammetry_importer.export_opengl_render_image

bl_label = Export Point Cloud Rendering as Image

bl_description = Use a single camera to render the point cloud.

filename_ext =

classmethod poll(*context*)

Return the availability status of the operator.

execute(*context*)

Render the point cloud and export the result as image.

class photogrammetry_importer.panels.render_operators.**ExportOpenGLRenderAnimationOperator**

Bases: bpy.types.Operator, bpy_extras.io_utils.ExportHelper

An Operator to save multiple renderings of the point cloud to disk.

bl_idname = photogrammetry_importer.export_opengl_render_animation

bl_label = Export Point Cloud Renderings as Image Sequence

bl_description = Use an animated camera to render the point cloud.

filename_ext =

classmethod poll(*context*)

Return the availability status of the operator.

execute(*context*)

Render the point cloud and export the result as image sequence.

photogrammetry_importer.panels.screenshot_operators

Module Contents

class photogrammetry_importer.panels.screenshot_operators.**ExportScreenshotImageOperator**

Bases: bpy.types.Operator, bpy_extras.io_utils.ExportHelper

An Operator to export a screenshot (of the 3D view).

bl_idname = photogrammetry_importer.export_screenshot

bl_label = Export Screenshot

bl_description = Create a screenshot (using a camera perspective).

filename_ext =

classmethod poll(*context*)

Return the availability status of the operator.

execute(*context*)

Export a screenshot (of the 3D view).

class photogrammetry_importer.panels.screenshot_operators.**ExportScreenshotAnimationOperator**

Bases: bpy.types.Operator, bpy_extras.io_utils.ExportHelper

An Operator to export a screenshot sequence (of the 3D view).

bl_idname = photogrammetry_importer.export_screenshot_sequence

bl_label = Export Screenshot Sequence

bl_description = Use the animation data to create a screenshot sequence.

filename_ext =

classmethod poll(*context*)

Return the availability status of the operator.

execute(*context*)

Export a sequence of screenshots using the selected camera.

photogrammetry_importer.panels.view_3d_panel

Module Contents

class photogrammetry_importer.panels.view_3d_panel.**OpenGLPanelSettings**

Bases: bpy.types.PropertyGroup

Class that defines the properties of the OpenGL panel in the 3D view.

viz_point_size :IntProperty(name='Point Size', description='OpenGL visualization point size.', get=get_viz_point_size, set=set_viz_point_size, min=1)

only_3d_view :BoolProperty(name='Export Only 3D View', description='Export only the 3D view or the full UI of Blender', default=True)

use_camera_perspective :BoolProperty(name='Use Perspective of Selected Camera', description='', default=True)

screenshot_file_format :StringProperty(name='File format', description='File format of the exported screenshot(s)', default='png')

use_camera_keyframes_for_screenshots :BoolProperty(name='Use Keyframes of Selected Camera', description='Use the Camera Keyframes instead of Animation Frames', default=True)

save_point_size :IntProperty(name='Point Size', description='OpenGL point size.', default=10)

render_file_format :StringProperty(name='File format', description='File format of the exported rendering(s)', default='png')

save_alpha :BoolProperty(name='Save Alpha Values', description='Save alpha values (if possible) to disk.', default=True)

use_camera_keyframes_for_rendering :BoolProperty(name='Use Camera Keyframes', description='Use the Camera Keyframes instead of Animation Frames.', default=True)

get_viz_point_size()

set_viz_point_size(*value*)

class photogrammetry_importer.panels.view_3d_panel.**OpenGLPanel**

Bases: bpy.types.Panel

Class that defines the OpenGL panel in the 3D view.

bl_label = OpenGL Panel

bl_idname = EXPORT_OPENGL_PT_render_point_cloud

bl_space_type = VIEW_3D

bl_region_type = UI

bl_category = PhotogrammetryImporter

classmethod poll(*context*)

Return the availability status of the panel.

classmethod register()

Register properties and operators corresponding to this panel.

classmethod unregister()

Unregister properties and operators corresponding to this panel.

draw(*context*)

Draw the panel with corrresponding properties and operators.

photogrammetry_importer.preferences

Contains persistent addon preferences.

Submodules

photogrammetry_importer.preferences.addon_preferences

Module Contents

class photogrammetry_importer.preferences.addon_preferences.**AddonPreferences**

Bases: bpy.types.AddonPreferences, *photogrammetry_importer.importers.camera_importer.CameraImporter*, *photogrammetry_importer.importers.point_importer.PointImporter*, *photogrammetry_importer.importers.mesh_importer.MeshImporter*

Class to manage persistent addon preferences.

bl_idname

visible_preferences :EnumProperty(name='Use original frames', items=(('DEPENDENCIES', 'Dependencies', ''), ('IMPORTEREXPORTER', 'Importer / Exporter', ''), ('IMPORTOPTIONS', 'Import Options', '')))

colmap_importer_bool :BoolProperty(name='Colmap Importer', default=True)

meshroom_importer_bool :BoolProperty(name='Meshroom Importer', default=True)

mve_importer_bool :BoolProperty(name='MVE Importer', default=True)

open3d_importer_bool :BoolProperty(name='Open3D Importer', default=True)

opensfm_importer_bool :BoolProperty(name='OpenSfM Importer', default=True)

openmvg_importer_bool :BoolProperty(name='OpenMVG Importer', default=True)

point_data_importer_bool :BoolProperty(name='Point Data Importer', default=True)

visualsfm_importer_bool :BoolProperty(name='VisualSfM Importer', default=True)

colmap_exporter_bool :BoolProperty(name='Colmap Exporter', default=True)

visualsfm_exporter_bool :BoolProperty(name='VisualSfM Exporter', default=True)

sys_path_list_str :StringProperty(name='System Path List Decoded String', default='[]')

classmethod register()

Register corresponding operators.

classmethod unregister()

Unregister corresponding operators.

draw(*context*)

Draw available preference options.

reset_import_options()

Reset the import options to factor settings.

class photogrammetry_importer.preferences.addon_preferences.**UpdateImporterExporterOperator**

Bases: bpy.types.Operator

Operator to activate and deactivate importers and exporters.

bl_idname = photogrammetry_importer.update_importer_exporter

bl_label = Update (Enable / Disable) Importers and Exporters

execute(*context*)

Activate and deactivate importers and exporters.

Uses the selected options of *AddonPreferences* to determine active and inactive importers and exporters.

class photogrammetry_importer.preferences.addon_preferences.**ResetImportOptionsOperator**

Bases: bpy.types.Operator

Operator to reset import options.

bl_idname = photogrammetry_importer.reset_import_options

bl_label = Reset Import Options to Factory Settings

execute(*context*)

Reset import options to factory settings.

photogrammetry_importer.preferences.dependency

Module Contents

photogrammetry_importer.preferences.dependency.**get_additional_command_line_sys_path**()

Function that retrieves additional sys.path of the command line

photogrammetry_importer.preferences.dependency.**add_command_line_sys_path**()

Function that adds sys.path of the command line to Blender's sys.path

photogrammetry_importer.preferences.dependency.**remove_command_line_sys_path**()

Function that removes additional paths in Blender's sys.path

photogrammetry_importer.preferences.dependency.**add_command_line_sys_path_if_necessary**(*dummy*)

Function that extends Blender's sys.path if necessary

class photogrammetry_importer.preferences.dependency.**DependencyStatus**(*gui_name*, *package_name*, *import_name*)

Class that describes the installation status of a Python dependency.

get_package_info()

class photogrammetry_importer.preferences.dependency.**PipManager**

Class that manages the pip installation.

classmethod get_singleton()

Return a singleton of this class.

install_pip(*lazy*, *op=None*)

Install pip.

get_package_info()

Return the pip installation status.

class photogrammetry_importer.preferences.dependency.**OptionalDependency**(*gui_name*, *package_name*, *import_name*)

Bases: *DependencyStatus*

Class that describes an optional Python dependency of the addon.

install(*op=None*)

Install this dependency.

uninstall(*remove_sys_path=True*, *op=None*)

Uninstall this dependency.

class photogrammetry_importer.preferences.dependency.**OptionalDependencyManager**

Class that manages the (optional) dependencies of this addon.

classmethod get_singleton()

Return a singleton of this class.

install_dependencies(*dependency_package_name=''*, *op=None*)

Install all (optional) dependencies of this addon.

uninstall_dependencies(*dependency_package_name=''*, *op=None*)

Uninstall all (optional) dependencies of this addon.

get_dependencies()

Return all (optional) dependencies of this addon.

class photogrammetry_importer.preferences.dependency.**InstallOptionalDependenciesOperator**

Bases: bpy.types.Operator

Operator to install all (optional) dependencies of this addon.

bl_idname = photogrammetry_importer.install_dependencies

bl_label = Download and Install ALL Optional Dependencies (be patient!)

bl_description = Download and install the optional dependencies (Python packages). Depending on the installation...

bl_options

dependency_package_name :StringProperty(name='Dependency Package Name', description='Target dependency package to be installed.', default='')

execute(*context*)

Install all optional dependencies.

class photogrammetry_importer.preferences.dependency.**UninstallOptionalDependenciesOperator**

Bases: bpy.types.Operator

Operator to uninstall all (optional) dependencies of this addon.

bl_idname = photogrammetry_importer.uninstall_dependencies

bl_label = Remove ALL Optional Dependencies

bl_description = Uninstall optional dependencies. Blender may have to be started with administrator privileges in...

bl_options

dependency_package_name :StringProperty(name='Dependency Package Name', description='Target dependency package to be removed.', default='')

execute(*context*)

Uninstall all optional dependencies.

photogrammetry_importer.registration

Contains functions to register import and export operators.

Submodules

photogrammetry_importer.registration.registration

Module Contents

class photogrammetry_importer.registration.registration.**Registration**

Class to register import and export operators.

classmethod **register_importers**(*import_prefs*)

Register importers according to the import preferences.

classmethod **unregister_importers**()

Unregister all registered importers.

classmethod **register_exporters**(*export_prefs*)

Register exporters according to the export preferences.

classmethod **unregister_exporters**()

Unregister all registered exporters.

photogrammetry_importer.types

Contains data types used to represent reconstruction results.

Submodules

photogrammetry_importer.types.camera

Module Contents

class photogrammetry_importer.types.camera.**Camera**

This class represents a reconstructed camera.

It provides functionality to manage intrinsic and extrinsic camera parameters as well as corresponding image and depth map information.

panoramic_type_equirectangular = EQUIRECTANGULAR

IMAGE_FP_TYPE_NAME = NAME

IMAGE_FP_TYPE_RELATIVE = RELATIVE

IMAGE_FP_TYPE_ABSOLUTE = ABSOLUTE

DEPTH_MAP_WRT_UNIT_VECTORS = DEPTH_MAP_WRT_UNIT_VECTORS

DEPTH_MAP_WRT_CANONICAL_VECTORS = DEPTH_MAP_WRT_CANONICAL_VECTORS

get_file_name()

Return the file name of the image used to register this camera.

set_relative_fp(*relative_fp*, *image_fp_type*)

Set the relative file path of the corresponding image.

get_relative_fp()

Return the relative file path of the corresponding image.

get_undistorted_relative_fp()

Return the relative file path of the undistorted image.

set_absolute_fp(*absolute_fp*)

Set the absolute file path of the corresponding image.

get_absolute_fp()

Return the absolute file path of the corresponding image.

get_undistorted_absolute_fp()

Return the absolute file path of the undistorted image.

has_undistorted_absolute_fp()

Determine if there is an absolute path to the undistorted image.

get_undistorted_file_name()

Return the file name of the undistorted image.

set_calibration(*calibration_mat*, *radial_distortion*)
Set calibration matrix and distortion parameter.

has_focal_length()
Return wether the focal length value has been defined or not.

get_focal_length()
Return the focal length value.

get_field_of_view()
Return the field of view corresponding to the focal length.

has_intrinsics()
Return wether the intrinsic parameters have been defined or not.

get_calibration_mat()
Return the calibration matrix.

set_calibration_mat(*calibration_mat*)
Set the calibration matrix.

set_principal_point(*principal_point*)
Set the principal point.

get_principal_point()
Return the principal point.

has_principal_point()
Return wether the principal point has been defined or not.

is_panoramic()
Return wether the camera model is a panoramic camera or not.

set_panoramic_type(*panoramic_type*)
Set the panoramic camera type.

get_panoramic_type()
Return the panoramic camera type (if any).

static compute_calibration_mat(*focal_length*, *cx*, *cy*)
Return the calibration matrix.

set_rotation_with_quaternion(*quaternion*)
Set the camera rotation using a quaternion.

set_rotation_with_rotation_mat(*rotation_mat*, *check_rotation=True*)
Set the camera rotation using a rotation matrix.

set_camera_center_after_rotation(*center*, *check_rotation=True*)
Set the camera center after setting the camera rotation.

set_camera_translation_vector_after_rotation(*translation_vector*, *check_rotation=True*)
Set the camera translation after setting the camera rotation.

get_rotation_as_quaternion()
Return the rotation as quaternion.

get_rotation_as_rotation_mat()
Return the rotation as rotation matrix.

get_translation_vec()

Return the translation vector.

get_camera_center()

Return the camera center.

set_4x4_cam_to_world_mat(*cam_to_world_mat*, *check_rotation=True*)

Set the extrinsic parameters.

static quaternion_to_rotation_matrix(*q*)

Convert a quaternion to a rotation matrix.

static rotation_matrix_to_quaternion(*m*)

Convert a rotation matrix to a quaternion.

set_depth_map_callback(*depth_map_callback*, *depth_map_ifp*, *depth_map_semantic*, *shift_depth_map_to_pixel_center*)

Set the depth map callback.

get_depth_map_fp()

Return the depth map file path.

get_depth_map()

Return the depth map.

get_4x4_cam_to_world_mat()

Return the camera to world transformation matrix.

This matrix can be used to convert homogeneous points given in camera coordinates to homogeneous points given in world coordinates.

convert_depth_map_to_world_coords(*depth_map_display_sparsity=100*)

Convert the depth map to points in world coordinates.

convert_cam_coords_to_world_coords(*cam_coords*)

Convert camera coordinates to world coordinates.

convert_depth_map_to_cam_coords(*depth_map_display_sparsity=100*)

Convert the depth map to points in camera coordinates.

photogrammetry_importer.types.point

Module Contents

class photogrammetry_importer.types.point.**Point**

Bases: `namedtuple('Point', ['coord', 'color', 'id', 'scalars'])`

This class represents a three-dimensional point.

A point contains the following attributes: 3D coordinate, color, point id and a list of scalars.

static split_points(*points*, *normalize_colors=False*)

Split points into coordinates and colors.

static create_points(*coords*, *colors*, *unnormalize_colors=False*)

static get_centered_points(*points*)

photogrammetry_importer.utility

Contains general utility functions.

Submodules

photogrammetry_importer.utility.developer_utility

Module Contents

photogrammetry_importer.utility.developer_utility.**setup_addon_modules**(*path*, *package_name*, *reload*)

Imports and reloads all modules in this addon.

Individual modules can define a `__reload_order_index__` property which will be used to reload the modules in a specific order. The default is 0.

photogrammetry_importer.utility.os_utility

Module Contents

photogrammetry_importer.utility.os_utility.**get_file_paths_in_dir**(*idp*, *ext=None*, *target_str_or_list=None*, *ignore_str_or_list=None*, *base_name_only=False*, *relative_path_only=False*, *without_ext=False*, *sort_result=True*, *natural_sorting=False*, *recursive=False*)

Return the paths of the files in the given directory.

The parameter `ext` can be a list of extensions or a single extension (e.g. [`.jpg`, `.png`] or `.jpg`).

photogrammetry_importer.utility.os_utility.**get_image_file_paths_in_dir**(*idp*, *base_name_only=False*, *relative_path_only=False*, *without_ext=False*, *sort_result=True*, *recursive=True*, *target_str_or_list=None*)

Return the paths of the images in the given directory.

photogrammetry_importer.utility.os_utility.**get_subdirs**(*idp*, *base_name_only=False*, *sort_result=True*, *natural_sorting=False*, *recursive=False*)

Return the paths of the subdirectories in the given directory.

photogrammetry_importer.utility.timing_utility

Module Contents

class photogrammetry_importer.utility.timing_utility.**StopWatch**

Bases: object

Class to measure computation times.

reset_time()

Reset the stop watch to the current point in time.

get_elapsed_time()

Return the elapsed time.

photogrammetry_importer.utility.type_utility

Module Contents

photogrammetry_importer.utility.type_utility.**is_int**(*some_str*)

Return True, if the given string represents an integer value.

photogrammetry_importer.utility.type_utility.**is_float**(*some_str*)

Return True, if the given string represents a float value.

photogrammetry_importer.utility.ui_utility

Module Contents

photogrammetry_importer.utility.ui_utility.**add_multi_line_label**(*ui_layout*, *long_text*, *max_line_length=120*)

Package Contents

photogrammetry_importer.**bl_info**

photogrammetry_importer.**modules**

photogrammetry_importer.**register**()

Register importers, exporters and panels.

photogrammetry_importer.**unregister**()

Unregister importers, exporters and panels.

There is a short tutorial video that shows how to

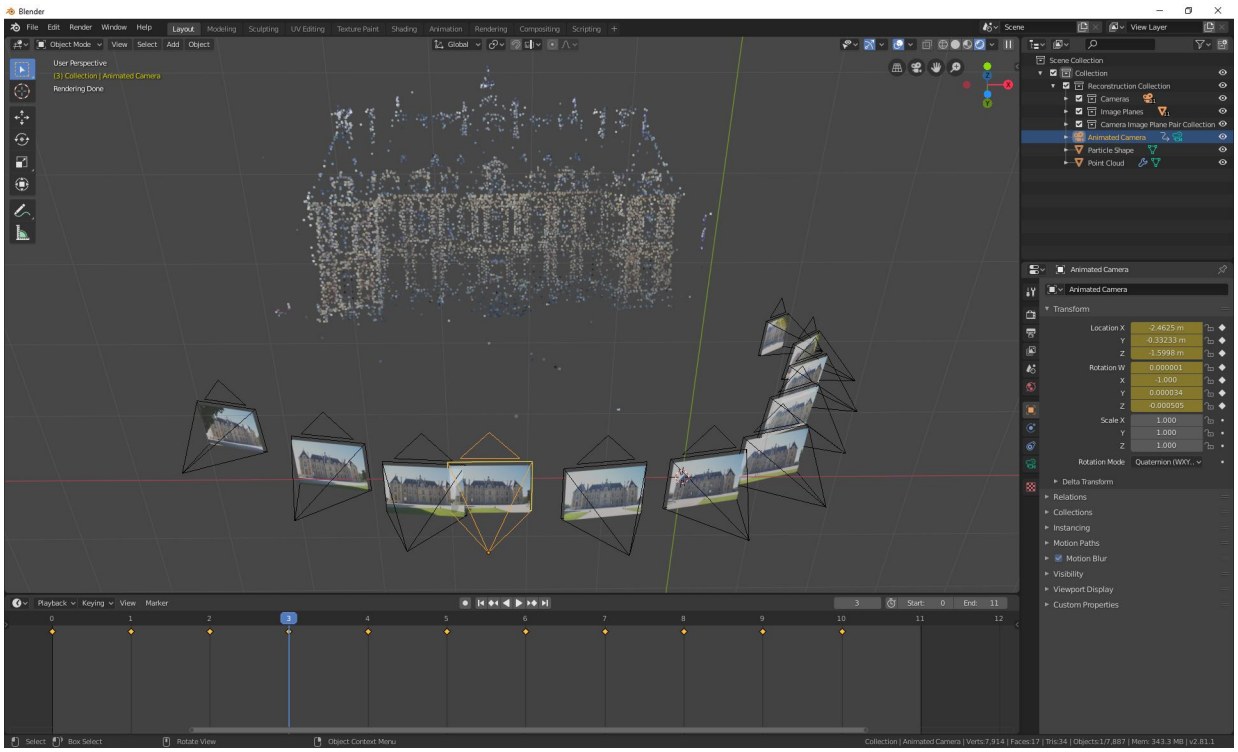
- install the addon
- compute a reconstruction with Meshroom
- import the results into Blender

CHAPTER

TWO

EXAMPLE RESULTS (SHIPPED WITH ADDON)

This repository contains an example Colmap model. The following image shows the imported camera poses, image planes and point cloud in Blender's 3D view.

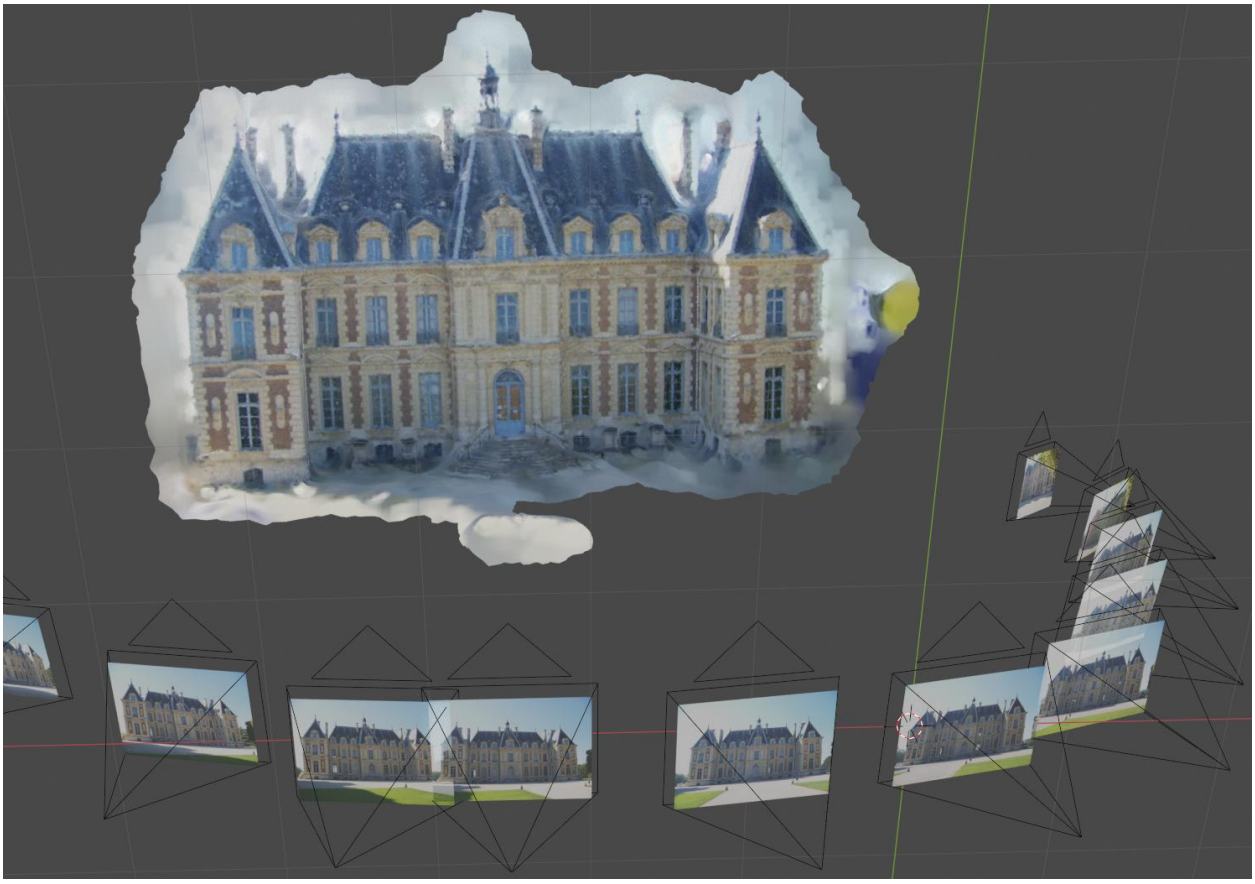


The input images of the Colmap model are located here: https://github.com/openMVG/ImageDataset_SceauxCastle.

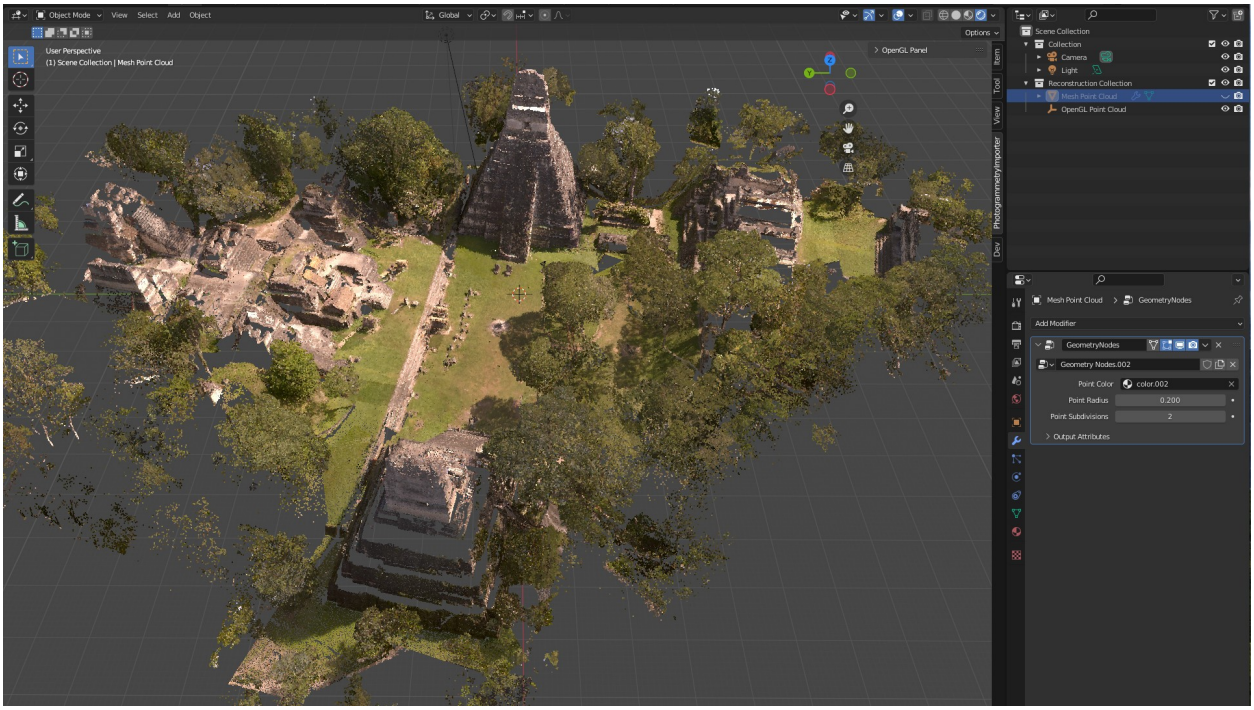
The addon computes an animated camera with corresponding background images from the reconstructed camera poses.

There is also an import option that allows to interpolate the reconstructed camera poses.

In addition, the addon allows to import meshes contained in the workspaces of specific libraries. Manually imported meshes can also be aligned with the corresponding reconstruction by following the instructions *here*.



The addon offers an option to draw big point clouds with OpenGL to reduce computational requirements. The addon provides a panel to export these OpenGL point clouds renderings - see *Point Cloud Visualization and Rendering*.



PYTHON MODULE INDEX

p

INDEX

A

B

D

E

H

I

Q

R

S

T

U

V

W