xiuminglib Documentation

Release 0.1

Xiuming Zhang

Jun 03, 2021

Contents:

xiuminglib includes daily classes and functions that are useful for my computer vision/graphics research. Noteworthily, it contains useful functions for 3D modeling and rendering with Blender.

To get a sense of what it is capable of, scroll to the bottom for a tree of its modules and functions. The source code is available in the repo. For issues or questions, please open an issue there.

CHAPTER 1

Installation

First, clone the repo. and add it to your `PYTHONPATH`:

```
cd <your_local_dir>
git clone https://github.com/xiumingzhang/xiuminglib.git
export PYTHONPATH="<your_local_dir>/xiuminglib/":"$PYTHONPATH"
```

Install the dependencies automatically with Conda: simply create an environment with all the dependencies by running:

```
cd <your_local_dir>/xiuminglib/
conda env create -f environment.yml
conda activate xiuminglib
```

If you do not need Blender functionalities, you are all set. Otherwise, you need to (manually) install Blender as a Python module, as instructed below.

If you want to avoid Conda environments, also see the dependencies below and manually install each your own way.

1.1 (Optional) Manual Dependency Installation

The library uses "on-demand" imports whenever possible, so that it will not fail on imports that you do not need.

If you want Blender, you need to install it as a Python module manually (regardless of using Conda or not):

Blender 2.79 Note this is different from installing Blender as an application, which has Python bundled. Rather, this is installing Blender as a Python module: you have succeeded if you find `bpy.so` in the build's bin folder and can `import bpy` in your Python (not the Blender-bundled Python) after you add it to your `PYTHONPATH`.

Ubuntu I did this "the hard way": first building all dependencies from source manually, and then building Blender from source with `-DWITH_PYTHON_MODULE=ON` for CMake, primarily because I wanted to build to an NFS location so that a cluster of machines on the NFS can all use the build.

If you only need Blender on a local machine, for which you can `sudo`, then dependency installations are almost automatic – just run `install_deps.sh`, although when I did this, I had to `skip-osl` to complete the run, for some reason I did not take time to find out.

Blender 2.80 made some API changes that are incompatible with this library, so please make sure after `git clone`, you check out the correct tag with `git checkout v2.79b`, followed by `git submodule update` to ensure the submodules are of the correct versions.

If `import bpy` throws `Segmentation fault`, try again with Python 3.6.3.

macOS This instruction was not very helpful, so below documents each step I took to finally get it working (though with some non-fatal warnings).

First, install Xcode 9.4 to build against the old `libstdc++` (instead of Xcode 10+ that forces the use of the newer `libc++`). Then, `brew install` CMake.

Install Python Framework 3.6.3. I tried to use an Anaconda Python, but to no avail.

Clone the Blender repo., check out v2.79b, and make sure submodules are consistent.

```
mkdir ~/blender-git && cd ~/blender-git
git clone https://git.blender.org/blender.git && cd blender
git checkout v2.79b # may also work: git reset --hard v2.79b
git submodule update --init --recursive
git submodule foreach git checkout master
git submodule foreach git pull --rebase origin master
```

Download the pre-built libraries, and move them to the correct place.

```
cd ~/blender-git
svn export https://svn.blender.org/svnroot/bf-blender/tags/blender-2.
↪79-release/lib/darwin-9.x.universal/
mkdir lib && mv darwin-9.x.universal lib/
```

Edit `~/blender-git/blender/build_files/cmake/platform/platform_apple.cmake` to replace `set(PYTHON_VERSION 3.5)` with `set(PYTHON_VERSION 3.6)`.

Make `bpy.so` by running `cd ~/blender-git/blender && make bpy`. You *may* also need `cd ~/blender-git/build_darwin_bpy && make install`. Upon success, `bpy.so` is in `~/blender-git/build_darwin_bpy/bin/`, and so is `2.79/`.

For `scripts/modules` to be found during import, do

```
mkdir ~/blender-git/build_darwin_bpy/Resources
cp -r ~/blender-git/build_darwin_bpy/bin/2.79 ~/blender-git/build_
↪darwin_bpy/Resources/
```

Add the bin folder to `PYTHONPATH` with `export PYTHONPATH="~/blender-git/build_darwin_bpy/bin/":"$PYTHONPATH"`.

Verify your success with

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 \
    -c 'import bpy; bpy.ops.render.render(write_still=True)'
```

but expect the aforementioned "non-fatal warnings":

```
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
```

(continues on next page)

(continued from previous page)

```
    mod = __import__(module_name)
ModuleNotFoundError: No module named 'io_scene_3ds'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
    mod = __import__(module_name)
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_scene_fbx/__init__.py", line 52, in <module>
    from bpy_extras.io_utils import (
ImportError: cannot import name 'orientation_helper'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
    mod = __import__(module_name)
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_anim_bvh/__init__.py", line 49, in <module>
    from bpy_extras.io_utils import (
ImportError: cannot import name 'orientation_helper'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
    mod = __import__(module_name)
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_mesh_ply/__init__.py", line 56, in <module>
    from bpy_extras.io_utils import (
ImportError: cannot import name 'orientation_helper'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
    mod = __import__(module_name)
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_scene_obj/__init__.py", line 48, in <module>
    from bpy_extras.io_utils import (
ImportError: cannot import name 'orientation_helper'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
    mod = __import__(module_name)
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_scene_x3d/__init__.py", line 48, in <module>
    from bpy_extras.io_utils import (
ImportError: cannot import name 'orientation_helper'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 331, in enable
    mod = __import__(module_name)
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_mesh_stl/__init__.py", line 66, in <module>
    from bpy_extras.io_utils import (
ImportError: cannot import name 'orientation_helper'
Exception in module register(): '/Users/xiuming/blender-git/build_
↪darwin_bpy/Resources/2.79/scripts/addons/io_curve_svg/__init__.py'
Traceback (most recent call last):
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/modules/addon_utils.py", line 350, in enable
    mod.register()
  File "/Users/xiuming/blender-git/build_darwin_bpy/Resources/2.79/
↪scripts/addons/io_curve_svg/__init__.py", line 70, in register
```

(continues on next page)

(continued from previous page)

```
    bpy.types.TOPBAR_MT_file_import.append(menu_func_import)
AttributeError: 'RNA_Types' object has no attribute 'TOPBAR_MT_file_
↪import'
```

Only if you are not automatically installing the dependencies, you need to manually install whatever you need:

NumPy The package for scientific computing that should be already available as part of your Python distribution.

SciPy The scientific computing ecosystem that may or may not be pre-installed already.

Matplotlib 2.0.2 Some functions are known to be buggy with 3.0.0.

tqdm A progress bar.

Pillow The friendly PIL fork.

OpenCV `pip install opencv-python` seems to work better than `conda install`. If any `lib*.so*` is missing at runtime (which happens often with `conda install`), the easiest fix is to install the missing library to the same environment, maybe followed by some symlinking (like linking `libjasper.so` to `libjasper.so.1`) inside `<python_dir>/envs/<env_name>/lib`. This may be cleaner and easier than `apt-get`, which may break other things and usually requires `sudo`.

Trimesh See their installation guide.

TensorFlow See this installation guide.

IPython This is required only for debugging purposes (e.g., inserting breakpoints with its `embed()`). Skip it if you do not care.

Sphinx 2.0.1 & RTD Theme These are required only by documentation building. Feel free to skip them if you do not care. The RTD theme package is called `sphinx_rtd_theme`.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

xiuminglib

3.1 xiuminglib package

3.1.1 Subpackages

xiuminglib.blender package

Submodules

xiuminglib.blender.camera module

xiuminglib.blender.camera.**add_camera**(*xyz=(0, 0, 0)*, *rot_vec_rad=(0, 0, 0)*, *name=None*, *proj_model='PERSP'*, *f=35*, *sensor_fit='HORIZONTAL'*, *sensor_width=32*, *sensor_height=18*, *clip_start=0.1*, *clip_end=100*)

Adds a camera to the current scene.

Parameters

- **xyz** (*tuple, optional*) – Location. Defaults to `(0, 0, 0)`.
- **rot_vec_rad** (*tuple, optional*) – Rotations in radians around x, y and z. Defaults to `(0, 0, 0)`.
- **name** (*str, optional*) – Camera object name.
- **proj_model** (*str, optional*) – Camera projection model. Must be `'PERSP'`, `'ORTHO'`, or `'PANO'`. Defaults to `'PERSP'`.
- **f** (*float, optional*) – Focal length in mm. Defaults to 35.
- **sensor_fit** (*str, optional*) – Sensor fit. Must be `'HORIZONTAL'` or `'VERTICAL'`. See also *get_camera_matrix()*. Defaults to `'HORIZONTAL'`.
- **sensor_width** (*float, optional*) – Sensor width in mm. Defaults to 32.

- **sensor_height** (*float, optional*) – Sensor height in mm. Defaults to 18.
- **clip_start** (*float, optional*) – Near clipping distance. Defaults to 0.1.
- **clip_end** (*float, optional*) – Far clipping distance. Defaults to 100.

Returns Camera added.

Return type bpy_types.Object

xiuminglib.blender.camera.**backproject_to_3d**(*xys*, *cam*, *obj_names=None*, *world_coords=False*)

Backprojects 2D coordinates to 3D.

Since a 2D point could have been projected from any point on a 3D line, this function will return the 3D point at which this line (ray) intersects with an object for the first time.

Parameters

- **xys** (*array_like*) – XY coordinates of length 2 or shape N-by-2, in the following convention:

```
(0, 0)
+------------> (w, 0)
|            x
|
|
|
v y (0, h)
```

- **cam** (*bpy_types.Object*) – Camera.
- **obj_names** (*str or list(str), optional*) – Name(s) of object(s) of interest. `None` means considering all objects.
- **world_coords** (*bool, optional*) – Whether to return world or the object's local coordinates.

Returns

- **ray_tos** (*mathutils.Vector or list(mathutils.Vector)*) – Location(s) at which each ray points in the world coordinates, regardless of `world_coords`. This and the (shared) ray origin (`cam.location`) determine the rays.
- **xyzs** (*mathutils.Vector or list(mathutils.Vector)*) – Intersection coordinates specified in either the world or the object's local coordinates, depending on `world_coords`. `None` means no intersection.
- **intersect_objnames** (*str or list(str)*) – Name(s) of object(s) responsible for intersections. `None` means no intersection.
- **intersect_facei** (*int or list(int)*) – Index/indices of the face(s), where the intersection happens.
- **intersect_normals** (*mathutils.Vector or list(mathutils.Vector)*) – Normal vector(s) at the intersection(s) specified in the same space as `xyzs`.

Return type tuple

xiuminglib.blender.camera.**correct_sensor_height**(*cam*)

To make render resolutions, sensor size, and pixel aspect ratio comptible.

If render resolutions are $(w_{\text{pix}}, h_{\text{pix}})$, sensor sizes are $(w_{\text{mm}}, h_{\text{mm}})$, and pixel aspect ratio is r, then $h_{\text{mm}} \leftarrow \frac{h_{\text{pix}}}{w_{\text{pix}} r} w_{\text{mm}}$.

Parameters **cam** (*bpy_types.Object*) – Camera.

xiuminglib.blender.camera.**easyset**(*cam*, *xyz=None*, *rot_vec_rad=None*, *name=None*, *proj_model=None*, *f=None*, *sensor_fit=None*, *sensor_width=None*, *sensor_height=None*)

Sets camera parameters more easily.

See *add_camera()* for arguments. `None` will result in no change.

xiuminglib.blender.camera.**get_2d_bounding_box**(*obj*, *cam*)

Gets a 2D bounding box of the object in the camera frame.

This is different from projecting the 3D bounding box to 2D.

Parameters

- **obj** (*bpy_types.Object*) – Object of interest.
- **cam** (*bpy_types.Object*) – Camera.

Returns

2D coordinates of the bounding box corners. Of shape 4-by-2. Corners are ordered counter-clockwise, following:

```
(0, 0)
+------------> (w, 0)
|            x
|
|
|
v y (0, h)
```

Return type numpy.ndarray

xiuminglib.blender.camera.**get_camera_matrix**(*cam*, *keep_disparity=False*)

Gets camera matrix, intrinsics, and extrinsics from a camera.

You can ask for a 4-by-4 projection that projects $(x, y, z, 1)$ to $(x, y, 1, d)$, where d is the disparity, reciprocal of depth.

`cam_mat.dot(pts)` gives you projections in the following convention:

```
+------------>
|       proj[:, 0]
|
|
v proj[:, 1]
```

Parameters

- **cam** (*bpy_types.Object*) – Camera.
- **keep_disparity** (*bool, optional*) – Whether or not the matrices keep disparity.

Returns

- **cam_mat** (*mathutils.Matrix*) – Camera matrix, product of intrinsics and extrinsics. 4-by-4 if `keep_disparity`; else, 3-by-4.
- **int_mat** (*mathutils.Matrix*) – Camera intrinsics. 4-by-4 if `keep_disparity`; else, 3-by-3.

- **ext_mat** (*mathutils.Matrix*) – Camera extrinsics. 4-by-4 if `keep_disparity`; else, 3-by-4.

Return type tuple

xiuminglib.blender.camera.**get_camera_zbuffer**(*cam*, *save_to=None*, *hide=None*)

Gets z-buffer of the camera.

Values are z components in camera-centered coordinate system, where

- x is horizontal;
- y is down (to align with the actual pixel coordinates);
- right-handed: positive z is look-at direction and means "in front of camera."

Origin is the camera center, not image plane (one focal length away from origin).

Parameters

- **cam** (*bpy_types.Object*) – Camera.
- **save_to** (*str, optional*) – Path to which the .exr z-buffer will be saved. `None` means don't save.
- **hide** (*str or list(str)*) – Names of objects to be hidden while rendering this camera's z-buffer.

Returns Camera z-buffer.

Return type numpy.ndarray

xiuminglib.blender.camera.**get_visible_vertices**(*cam*, *obj*, *ignore_occlusion=False*, *hide=None*, *method='raycast'*, *perc_eps=1e-06*)

Gets vertices that are visible (projected within frame *and* unoccluded) from camera.

Parameters

- **cam** (*bpy_types.Object*) – Camera.
- **obj** (*bpy_types.Object*) – Object of interest.
- **ignore_occlusion** (*bool, optional*) – Whether to ignore all occlusion (including self-occlusion). Useful for finding out which vertices fall inside the camera view.
- **hide** (*str or list(str), optional*) – Names of objects to be hidden while rendering this camera's z-buffer. No effect if `ignore_occlusion`.
- **method** (*str, optional*) – Visibility test method: `'raycast'` or `'zbuffer'`. Ray casting is more robust than comparing the vertex's depth against z-buffer (inaccurate when the render resolution is low, or when object's own depth variation is small compared with its overall depth). The advantage of the z-buffer, though, is its runtime independent of number of vertices.
- **perc_eps** (*float, optional*) – Threshold for percentage difference between test value x and true value y. x is considered equal to y when $\frac{|x-y|}{y}$ is smaller. No effect if `ignore_occlusion`.

Returns Indices of vertices that are visible.

Return type list

xiuminglib.blender.camera.**intrinsics_compatible_with_scene**(*cam*, *eps=1e-06*)

Checks if camera intrinsic parameters are comptible with the current scene.

Intrinsic parameters include sensor size and pixel aspect ratio, and scene parameters refer to render resolutions and their scale. The entire sensor is assumed active.

Parameters

- **cam** (*bpy_types.Object*) – Camera object
- **eps** (*float, optional*) – ϵ for numerical comparison. Considered equal if $\frac{|a-b|}{b} < \epsilon$.

Returns Check result.

Return type bool

xiuminglib.blender.camera.**point_camera_to**(*cam*, *xyz_target*, *up=(0, 0, 1)*)

Points camera to target.

Parameters

- **cam** (*bpy_types.Object*) – Camera object.
- **xyz_target** (*array_like*) – Target point in world coordinates.
- **up** (*array_like, optional*) – World vector that, when projected, points up in the image plane.

xiuminglib.blender.light module

Add functions in this module usually provide no setter for the lamp's 3D rotation, because one usually implicitly sets the rotation by pointing the light to an object (and specifying an up vector), by using *point_light_to()*.

xiuminglib.blender.light.**add_light_area**(*xyz=(0, 0, 0)*, *rot_vec_rad=(0, 0, 0)*, *name=None*, *energy=100*, *size=0.1*)

Adds an area light that emits light rays the lambertian way.

Parameters

- **xyz** (*tuple(float), optional*) – Location.
- **rot_vec_rad** (*tuple(float), optional*) – Rotations in radians around x, y and z.
- **name** (*str, optional*) – Light name.
- **energy** (*float, optional*) – Light intensity.
- **size** (*float, optional*) – Light size for ray shadow tracing. Use larger values for softer shadows.

Returns Light added.

Return type bpy_types.Object

xiuminglib.blender.light.**add_light_env**(*env=(1, 1, 1, 1)*, *strength=1*, *rot_vec_rad=(0, 0, 0)*, *scale=(1, 1, 1)*)

Adds environment lighting.

Parameters

- **env** (*tuple(float) or str, optional*) – Environment map. If tuple, it's RGB or RGBA, each element of which $\in [0, 1]$. Otherwise, it's the path to an image.
- **strength** (*float, optional*) – Light intensity.
- **rot_vec_rad** (*tuple(float), optional*) – Rotations in radians around x, y and z.

- **scale** (*tuple(float), optional*) – If all changed simultaneously, then no effects.

xiuminglib.blender.light.**add_light_point**(*xyz=(0, 0, 0)*, *name=None*, *size=0*, *energy=100*)
Adds an omnidirectional point lamp.

Parameters

- **xyz** (*tuple(float), optional*) – Location.
- **name** (*str, optional*) – Light name.
- **size** (*float, optional*) – Light size; the larger the softer shadows are.
- **energy** (*float, optional*) – Light intensity.

Returns Light added.

Return type bpy_types.Object

xiuminglib.blender.light.**add_light_spot**(*xyz=(0, 0, 0)*, *name=None*, *energy=100*, *shadow_soft_size=0.1*, *spot_size=0.785*, *spot_blend=0.15*)
Adds a spotlight lamp.

Parameters

- **xyz** (*tuple(float), optional*) – Location.
- **name** (*str, optional*) – Light name.
- **energy** (*float, optional*) – Light intensity.
- **shadow_soft_size** (*float, optional*) – Light size for raytracing the shadow.
- **spot_size** (*float, optional*) – Angle, in radians, of the spotlight beam.
- **spot_blend** (*float, optional*) – Softness of the spotlight edge.

Returns Light added.

Return type bpy_types.Object

xiuminglib.blender.light.**add_light_sun**(*xyz=(0, 0, 0)*, *rot_vec_rad=(0, 0, 0)*, *name=None*, *energy=1*, *size=0.1*)
Adds a sun lamp that emits parallel light rays.

Parameters

- **xyz** (*tuple(float), optional*) – Location only used to compute light ray direction.
- **rot_vec_rad** (*tuple(float), optional*) – Rotations in radians around x, y and z.
- **name** (*str, optional*) – Light name.
- **energy** (*float, optional*) – Light intensity.
- **size** (*float, optional*) – Light size for ray shadow tracing. Use larger for softer shadows.

Returns Light added.

Return type bpy_types.Object

xiuminglib.blender.light.**point_light_to**(*light*, *target*)
Points the directional light to a target.

Parameters

- **light** (*bpy_types.Object*) – Light object.
- **target** (*tuple(float)*) – Target location to which light rays point.

xiuminglib.blender.object module

xiuminglib.blender.object.**add_cylinder_between**(*pt1*, *pt2*, *r=0.001*, *name=None*)

Adds a cylinder specified by two end points and radius.

Super useful for visualizing rays in ray tracing while debugging.

Parameters

- **pt1** (*array_like*) – World coordinates of point 1.
- **pt2** (*array_like*) – World coordinates of point 2.
- **r** (*float, optional*) – Cylinder radius.
- **name** (*str, optional*) – Cylinder name.

Returns Cylinder added.

Return type bpy_types.Object

xiuminglib.blender.object.**add_rectangular_plane**(*center_loc=(0, 0, 0)*, *point_to=(0, 0, 1)*, *size=(2, 2)*, *name=None*)

Adds a rectangular plane specified by its center location, dimensions, and where its +z points to.

Parameters

- **center_loc** (*array_like, optional*) – Plane center location in world coordinates.
- **point_to** (*array_like, optional*) – Point in world coordinates to which plane's +z points.
- **size** (*array_like, optional*) – Sizes in x and y directions (0 in z).
- **name** (*str, optional*) – Plane name.

Returns Plane added.

Return type bpy_types.Object

xiuminglib.blender.object.**add_sphere**(*location=(0, 0, 0)*, *scale=1*, *n_subdiv=2*, *shade_smooth=False*, *name=None*)

Adds a sphere.

Parameters

- **location** (*array_like, optional*) – Location of the sphere center.
- **scale** (*float, optional*) – Scale of the sphere.
- **n_subdiv** (*int, optional*) – Control of how round the sphere is.
- **shade_smooth** (*bool, optional*) – Whether to use smooth shading.
- **name** (*str, optional*) – Name of the added sphere.

Returns Sphere created.

Return type bpy_types.Object

xiuminglib.blender.object.**color_vertices**(*obj*, *vert_ind*, *colors*)
Colors each vertex of interest with the given color.

Colors are defined for vertex loops, in fact. This function uses the same color for all loops of a vertex. Useful for making a 3D heatmap.

Parameters

- **obj** (*bpy_types.Object*) – Object.
- **vert_ind** (*int or list(int)*) – Index/indices of vertex/vertices to color.
- **colors** (*tuple or list(tuple)*) – RGB value(s) to paint on vertex/vertices. Values $\in [0, 1]$. If one tuple, this color will be applied to all vertices. If list of tuples, must be of the same length as `vert_ind`.

xiuminglib.blender.object.**create_mesh**(*verts*, *faces*, *name='new-mesh'*)
Creates a mesh from vertices and faces.

Parameters

- **verts** (*array_like*) – Local coordinates of the vertices, of shape N-by-3.
- **faces** (*list(tuple)*) – Faces specified by ordered vertex indices.
- **name** (*str, optional*) – Mesh name.

Returns Mesh data created.

Return type bpy_types.Mesh

xiuminglib.blender.object.**create_object_from_mesh**(*mesh_data*, *obj_name='new-obj'*, *location=(0, 0, 0)*, *rotation_euler=(0, 0, 0)*, *scale=(1, 1, 1)*)
Creates object from mesh data.

Parameters

- **mesh_data** (*bpy_types.Mesh*) – Mesh data.
- **obj_name** (*str, optional*) – Object name.
- **location** (*tuple, optional*) – Object location in world coordinates.
- **rotation_euler** (*tuple, optional*) – Object rotation in radians.
- **scale** (*tuple, optional*) – Object scale.

Returns Object created.

Return type bpy_types.Object

xiuminglib.blender.object.**export_object**(*obj_names*, *model_path*, *axis_forward=None*, *axis_up=None*)
Exports Blender object(s) to a file.

Parameters

- **obj_names** (*str or list(str)*) – Object name(s) to export. Must be a single string if output format is .ply.
- **model_path** (*str*) – Output .obj or .ply path.
- **axis_forward** (*str, optional*) – Which direction is forward. For .obj, the default is `'-Z'`, and `'Y'` for .ply.

- **axis_up** (*str, optional*) – Which direction is upward. For .obj, the default is `'Y'`, and `'Z'` for .ply.

Writes

- Exported model file, possibly accompanied by a material file.

xiuminglib.blender.object.**get_bmesh**(*obj*)
Gets Blender mesh data from object.

Parameters **obj** (*bpy_types.Object*) – Object.

Returns Blender mesh data.

Return type BMesh

xiuminglib.blender.object.**get_object**(*otype*, *any_ok=False*)
Gets the handle of the only (or any) object of the given type.

Parameters

- **otype** (*str*) – Object type: `'MESH'`, `'CAMERA'`, `'LAMP'` or any string `a_bpy_obj.type` may return.
- **any_ok** (*bool, optional*) – Whether it's ok to grab any object when there exist multiple ones matching the given type. If `False`, there must be exactly one object of the given type.

Returns bpy_types.Object.

xiuminglib.blender.object.**import_object**(*model_path*, *axis_forward='-Z'*, *axis_up='Y'*, *rot_mat=((1, 0, 0), (0, 1, 0), (0, 0, 1))*, *trans_vec=(0, 0, 0)*, *scale=1*, *merge=False*, *name=None*)
Imports external object to current scene, the low-level way.

Parameters

- **model_path** (*str*) – Path to object to add.
- **axis_forward** (*str, optional*) – Which direction is forward.
- **axis_up** (*str, optional*) – Which direction is upward.
- **rot_mat** (*array_like, optional*) – 3-by-3 rotation matrix *preceding* translation.
- **trans_vec** (*array_like, optional*) – 3D translation vector *following* rotation.
- **scale** (*float, optional*) – Scale of the object.
- **merge** (*bool, optional*) – Whether to merge objects into one.
- **name** (*str, optional*) – Object name after import.

Returns Imported object(s).

Return type bpy_types.Object or list(bpy_types.Object)

xiuminglib.blender.object.**raycast**(*obj_bvhtree*, *ray_from_objspc*, *ray_to_objspc*)
Casts a ray to an object.

Parameters

- **obj_bvhtree** (*mathutils.bvhtree.BVHTree*) – Constructed BVH tree of the object.
- **ray_from_objspc** (*mathutils.Vector*) – Ray origin, in object's local coordinates.

- **ray_to_objspc** (*mathutils.Vector*) – Ray goes through this point, also specified in the object's local coordinates. Note that the ray doesn't stop at this point, and this is just for computing the ray direction.

Returns

- **hit_loc** (*mathutils.Vector*) – Hit location on the object, in the object's local coordinates. `None` means no intersection.
- **hit_normal** (*mathutils.Vector*) – Normal of the hit location, also in the object's local coordinates.
- **hit_fi** (*int*) – Index of the face where the hit happens.
- **ray_dist** (*float*) – Distance that the ray has traveled before hitting the object. If `ray_to_objspc` is a point on the object surface, then this return value is useful for checking for self occlusion.

Return type tuple

xiuminglib.blender.object.**remove_objects**(*name_pattern*, *regex=False*)
Removes object(s) from current scene.

Parameters

- **name_pattern** (*str*) – Name or name pattern of object(s) to remove.
- **regex** (*bool, optional*) – Whether to interpret `name_pattern` as a regex.

xiuminglib.blender.object.**select_mesh_elements_by_vertices**(*obj*, *vert_ind*, *select_type*)
Selects vertices or their associated edges/faces in edit mode.

Parameters

- **obj** (*bpy_types.Object*) – Object.
- **vert_ind** (*int or list(int)*) – Vertex index/indices.
- **select_type** (*str*) – Type of mesh elements to select: `'vertex'`, `'edge'` or `'face'`.

xiuminglib.blender.object.**setup_emission_nodetree**(*obj*, *texture=(1, 1, 1, 1)*, *strength=1*, *hide=False*)
Sets up an emission node tree for the object.

Parameters

- **obj** (*bpy_types.Object*) – Object (maybe bundled with texture map).
- **texture** (*str or tuple, optional*) – If string, must be `'bundled'` or path to the texture image. If tuple, must be of 4 floats $\in [0, 1]$ as RGBA values.
- **strength** (*float, optional*) – Emission strength.
- **hide** (*bool, optional*) – Useful for hiding the emissive object (but keeping the light of course).

xiuminglib.blender.object.**setup_holdout_nodetree**(*obj*)
Sets up a holdout node tree for the object.

Parameters **obj** (*bpy_types.Object*) – Object bundled with texture map.

xiuminglib.blender.object.**setup_retroreflective_nodetree**(*obj*, *texture*, *roughness=0*, *glossy_weight=0.1*)

Sets up a retroreflective texture node tree.

Bundled texture can be an external texture map (carelessly mapped) or a pure color. Mathematically, the BRDF model is a mixture of a diffuse BRDF and a glossy BRDF using incoming light directions as normals.

Parameters

- **obj** (*bpy_types.Object*) – Object, optionally bundled with texture map.
- **texture** (*str or tuple*) – If string, must be `'bundled'` or path to the texture image. If tuple, must be of 4 floats $\in [0, 1]$ as RGBA values.
- **roughness** (*float, optional*) – Roughness for both the glossy and diffuse shaders.
- **glossy_weight** (*float, optional*) – Mixture weight for the glossy shader.

xiuminglib.blender.object.**setup_simple_nodetree**(*obj*, *texture*, *shader_type*, *roughness=0*)

Sets up a simple (diffuse and/or glossy) node tree.

Texture can be an bundled texture map, a path to an external texture map, or simply a pure color. If a path to an external image, and UV coordinates are given (e.g., in the geometry .obj file), then they will be used. If they are not given, texture mapping will be done carelessly, with automatically generated UV coordinates. See private function `_make_texture_node()` for how this is done.

Parameters

- **obj** (*bpy_types.Object*) – Object, optionally bundled with texture map.
- **texture** (*str or tuple*) – If string, must be `'bundled'` or path to the texture image. If tuple, must be of 4 floats $\in [0, 1]$ as RGBA values.
- **shader_type** (*str*) – Either `'diffuse'` or `'glossy'`.
- **roughness** (*float, optional*) – If diffuse, the roughness in Oren-Nayar, 0 gives Lambertian. If glossy, 0 means perfectly reflective.

xiuminglib.blender.object.**smart_uv_unwrap**(*obj*, *area_weight=0.0*)

UV unwrapping using Blender's smart projection.

A vertex may map to multiple UV locations, but each loop maps to exactly one UV location. If a face uses M vertices, then it has M loops, so a vertex may belong to multiple loops, each of which has one UV location.

Note: If a vertex belongs to no face, it doesn't get a UV coordinate, so don't assume you can get a UV for any given vertex index.

Parameters

- **obj** (*bpy_types.Object*) – Object to UV unwrap.
- **area_weight** (*float, optional*) – Area weight.

Returns

Dictionary with its keys being the face indices, and values being 2D arrays with four columns containing the corresponding face's loop indices, vertex indices, u, and v.

UV coordinate convention:

```
(0, 1)
    ^ v
    |
    |
    |
    |
    +-----------> (1, 0)
(0, 0)        u
```

Return type dict(numpy.ndarray)

xiuminglib.blender.object.**subdivide_mesh**(*obj*, *n_subdiv=2*)
Subdivides mesh of object.

Parameters

- **obj** (*bpy_types.Object*) – Object whose mesh is to be subdivided.
- **n_subdiv** (*int, optional*) – Number of subdivision levels.

xiuminglib.blender.render module

xiuminglib.blender.render.**easyset**(*w=None*, *h=None*, *n_samples=None*, *ao=None*, *color_mode=None*, *file_format=None*, *color_depth=None*, *sampling_method=None*, *n_aa_samples=None*)
Sets some of the scene attributes more easily.

Parameters

- **w** (*int, optional*) – Width of render in pixels.
- **h** (*int, optional*) – Height of render in pixels.
- **n_samples** (*int, optional*) – Number of samples.
- **ao** (*bool, optional*) – Ambient occlusion.
- **color_mode** (*str, optional*) – Color mode of rendering: `'BW'`, `'RGB'`, or `'RGBA'`.
- **file_format** (*str, optional*) – File format of the render: `'PNG'`, `'OPEN_EXR'`, etc.
- **color_depth** (*str, optional*) – Color depth of rendering: `'8'` or `'16'` for .png; `'16'` or `'32'` for .exr.
- **sampling_method** (*str, optional*) – Method to sample light and materials: `'PATH'` or `'BRANCHED_PATH'`.
- **n_aa_samples** (*int, optional*) – Number of anti-aliasing samples (used with `'BRANCHED_PATH'`).

xiuminglib.blender.render.**render**(*outpath*, *cam=None*, *obj_names=None*, *alpha=True*, *text=None*)
Renders current scene with cameras in scene.

Parameters

- **outpath** (*str*) – Path to save the render to. Should end with either .exr or .png.
- **cam** (*bpy_types.Object, optional*) – Camera through which scene is rendered. If `None`, use the only camera in scene.

- **obj_names** (*str or list(str), optional*) – Name(s) of object(s) of interest. If `None`, all objects are of interest and will appear in the render.
- **alpha** (*bool, optional*) – Whether to render the alpha channel.
- **text** (*dict, optional*) – What text to be overlaid on image and how, following the format:

```
{
    'contents': "Hello World!",
    'bottom_left_corner': (50, 50),
    'font_scale': 1,
    'bgr': (255, 0, 0),
    'thickness': 2,
}
```

Writes

- A 32-bit .exr or 16-bit .png image.

xiuminglib.blender.render.**render_alpha**(*outpath*, *cam=None*, *obj_names=None*, *samples=1000*)

Renders binary or soft mask of objects from the specified camera.

Parameters

- **outpath** (*str*) – Path to save the render to. Should end with .png.
- **cam** (*bpy_types.Object, optional*) – Camera through which scene is rendered. If `None`, there must be just one camera in scene.
- **obj_names** (*str or list(str), optional*) – Name(s) of object(s) of interest. `None` means all objects.
- **samples** (*int, optional*) – Samples per pixel. 1 gives a hard mask, and 1 gives a soft (anti-aliased) mask.

Writes

- A 16-bit three-channel .png mask, where bright indicates foreground.

xiuminglib.blender.render.**render_depth**(*outprefix*, *cam=None*, *obj_names=None*, *ray_depth=False*)

Renders raw depth map in .exr of the specified object(s) from the specified camera.

The EXR data contain an aliased z map and an anti-aliased alpha map.

Parameters

- **outprefix** (*str*) – Where to save the .exr maps to, e.g., `'~/depth'`.
- **cam** (*bpy_types.Object, optional*) – Camera through which scene is rendered. If `None`, there must be the just one camera in the scene.
- **obj_names** (*str or list(str), optional*) – Name(s) of object(s) of interest. `None` means all objects.
- **ray_depth** (*bool, optional*) – Whether to render ray or plane depth.

Writes

- A 32-bit .exr depth map w/o anti-aliasing, located at `outprefix + '_z.exr'`.

- A 32-bit .exr alpha map w/ anti-aliasing, located at `outprefix + '_a.exr'`.

Todo: Ray depth.

xiuminglib.blender.render.**render_lighting_passes**(*outpath*, *cam=None*, *obj_names=None*, *n_samples=None*, *select=None*)

Renders select Cycles' lighting passes of the specified object(s) from the specified camera.

Data are in a single multi-layer .exr file. See the code below for what channels are rendered.

Parameters

- **outpath** (*str*) – Where to save the lighting passes to. Should end with .exr.
- **cam** (*bpy_types.Object, optional*) – Camera through which scene is rendered. If `None`, there must be only one camera in scene.
- **obj_names** (*str or list(str), optional*) – Name(s) of object(s) of interest. `None` means all objects.
- **n_samples** (*int, optional*) – Number of samples per pixel. Useful when you want a value different than other renderings; `None` means using the current value.
- **select** (*list(str), optional*) – Render only this list of passes. `None` means rendering all passes: `diffuse_direct`, `diffuse_indirect`, `diffuse_color`, `glossy_direct`, `glossy_indirect`, and `glossy_color`.

Writes

- A 32-bit .exr multi-layer image containing the lighting passes.

xiuminglib.blender.render.**render_normal**(*outpath*, *cam=None*, *obj_names=None*, *outpath_refball=None*, *world_coords=False*)

Renders raw normal map in .exr of the specified object(s) from the specified camera.

RGB at each pixel is the (almost unit) normal vector at that location.

Parameters

- **outpath** (*str*) – The .exr path (so data are raw values, not integer values) we save the normal map to.
- **cam** (*bpy_types.Object, optional*) – Camera through which scene is rendered. If `None`, there must be only one camera in scene.
- **obj_names** (*str or list(str), optional*) – Name(s) of object(s) of interest. `None` means all objects.
- **outpath_refball** (*str, optional*) – The .exr path to save the reference ball's normals to. `None` means not rendering the reference ball.
- **world_coords** (*bool, optional*) – Whether to render normals in the world or *negated* camera space.

Warning: TL;DR

If you want camera-space normal maps, you need to negate the normal map after loading it from the .exr file this function writes. Otherwise, those normals live in a space that has all three axes flipped w.r.t. the camera's local space.

The Details

If you want world-space normals, then easy; I'll just use Cycles, and the normals are automatically in the world space. If camera-space normals are what you want, I'll use Blender Internal (BI), but there's some complication taken care of under the hood.

BI renders normals "in the camera space." I verified this by keeping my scene intact, but having my camera rotating a bit; indeed, the normal vectors of a cube went from "round values", such as $(0, 0, 1)$, to "non-round values", such as $(0.02, 0.03, 0.99)$.

But what precisely is this "camera space" (denoted by S)? Is it really just the camera's local space? How do we go from S to the world coordinate system, and possibly to another space therefrom? Here's my exploration.

I put a camera at the scene center, and had it pointing, head-on, to one face of a default cube (so the camera saw just that face – no other faces). I rendered the normals with BI: the raw RGB value is $(0, 0, 1)$. OK, so the normal vector pointing out of the screen to my eyes is S's z. Hence, S's $+z$ points into the screen.

Then I rotated the cube by just a little, so my camera got to see a little bit of the side faces it couldn't see before. The normal vector pointing to the left is $(1, 0, 0)$; so the $+x$ of S points to the left, tangent to the screen. Similarly, I found out the $+y$ points downwards, also tangent to the screen.

But wait, the three axes don't even form a right-handed system; they form a left-handed one! This is so strange. Oh, if we negate all the axes, then we get a right-handed system. Would this negated system be the camera's local space (as in an object's local coordinate system)?

It is! After eyeballing the camera's local space axes, I found they are exactly the flipped axes of S. Therefore, when camera-space normals are asked for, I first render them out using BI, and then have to flip the signs to give the camera-space normals, which you can then transform to other spaces correctly with transformation matrices.

Writes

- A 32-bit .exr normal map of the object(s) of interest.
- Another 32-bit .exr normal map of the reference ball, if asked for.

`xiuminglib.blender.render.`**`set_cycles`**(*w=None*, *h=None*, *n_samples=None*, *max_bounces=None*, *min_bounces=None*, *transp_bg=None*, *color_mode=None*, *color_depth=None*)

Sets up Cycles as rendering engine.

`None` means no change.

Parameters

- **`w`** (*`int, optional`*) – Width of render in pixels.
- **`h`** (*`int, optional`*) – Height of render in pixels.
- **`n_samples`** (*`int, optional`*) – Number of samples.
- **`max_bounces`** (*`int, optional`*) – Maximum number of light bounces. Setting max_bounces to 0 for direct lighting only.
- **`min_bounces`** (*`int, optional`*) – Minimum number of light bounces.
- **`transp_bg`** (*`bool, optional`*) – Whether world background is transparent.
- **`color_mode`** (*`str, optional`*) – Color mode: `'BW'`, `'RGB'` or `'RGBA'`.

- **color_depth** (*str, optional*) – Color depth: `'8'` or `'16'`.

xiuminglib.blender.scene module

xiuminglib.blender.scene.**open_blend**(*inpath*)
Opens a .blend file.

Parameters **inpath** (*str*) – E.g., `'~/foo.blend'`.

xiuminglib.blender.scene.**save_blend**(*outpath=None*, *delete_overwritten=False*)
Saves current scene to a .blend file.

Parameters

- **outpath** (*str, optional*) – Path to save the scene to, e.g., `'~/foo.blend'`. `None` means saving to the current file.
- **delete_overwritten** (*bool, optional*) – Whether to delete or keep as .blend1 the same-name file.

Writes

- A .blend file.

xiuminglib.blender.util module

xiuminglib.blender.util.**cursor_to**(*loc*)
Moves the cursor to the given 3D location.

Useful for inspecting where a 3D point is in the scene, to do which you first use this function, save the scene, and open the scene in GUI.

Parameters **loc** (*array_like*) – 3D coordinates, of length 3.

xiuminglib.geometry package

Submodules

xiuminglib.geometry.depth module

xiuminglib.geometry.normal module

xiuminglib.geometry.normal.**gen_world2local**(*normal*)
Generates rotation matrices that transform world normals to local $+z$, world tangents to local $+x$, and world binormals to local $+y$.

Parameters **normal** (*numpy.ndarray*) – any size-by-3 array of normal vectors.

Returns Any size-by-3-by-3 world-to-local rotation matrices, which should be left-multiplied to world coordinates.

Return type numpy.ndarray

xiuminglib.geometry.normal.**normalize**(*normal_map*, *norm_thres=0.5*)
Normalizes the normal vector at each pixel of the normal map.

Parameters

- **normal_map** (*numpy.ndarray*) – H-by-W-by-3 array of normal vectors.
- **norm_thres** (*float, optional*) – Normalize only vectors with a norm greater than this; helpful to avoid errors at the boundary or in the background.

Returns Normalized normal map.

Return type numpy.ndarray

xiuminglib.geometry.normal.**transform_space**(*normal_map*, *rotmat*)
Transforms the normal vectors from one space to another.

Parameters

- **normal_map** (*numpy.ndarray*) – H-by-W-by-3 array of normal vectors.
- **rotmat** (*numpy.ndarray or mathutils.Matrix*) – 3-by-3 rotation matrix, which is left-multiplied to the vectors.

Returns Transformed normal map.

Return type numpy.ndarray

xiuminglib.geometry.proj module

xiuminglib.geometry.proj.**from_homo**(*pts*, *axis=None*)
Converts from homogeneous to non-homogeneous coordinates.

Parameters

- **pts** (*numpy.ndarray or mathutils.Vector*) – NumPy array of N-D point(s), or Blender vector of a single N-D point.
- **axis** (*int, optional*) – The last slice of which dimension holds the w values. Optional for 1D inputs.

Returns Non-homogeneous coordinates of the input point(s).

Return type numpy.ndarray or mathutils.Vector

xiuminglib.geometry.proj.**to_homo**(*pts*)
Pads 2/3D points to homogeneous, by guessing which dimension to pad.

Parameters **pts** (*array_like*) – Input array of 2D or 3D points.

Returns Homogeneous coordinates of the input points.

Return type numpy.ndarray

xiuminglib.geometry.pt module

xiuminglib.geometry.pt.**ptcld2tdf**(*pts*, *res=128*, *center=False*)
Converts point cloud to truncated distance function (TDF).

Maximum distance is capped at 1 / `res`.

Parameters

- **pts** (*array_like*) – Cartesian coordinates in object space. Of shape N-by-3.
- **res** (*int, optional*) – Resolution of the TDF.

- **center** (*bool, optional*) – Whether to center these points around the object space origin.

Returns Output TDF.

Return type numpy.ndarray

xiuminglib.geometry.rot module

xiuminglib.geometry.rot.**axis_angle_to_rot_mat**(*axis*, *theta*)
Gets rotation matrix that rotates points around an arbitrary axis by any angle.

Rotating around the x/y/z axis are special cases of this, where you simply specify the axis to be one of those axes.

Parameters

- **axis** (*array_like*) – 3-vector that specifies the end point of the rotation axis (start point is the origin). This will be normalized to be unit-length.
- **theta** (*float*) – Angle in radians, prescribed by the right-hand rule, so a negative value means flipping the rotation axis.

Returns 3×3 rotation matrix, to be pre-multiplied with the vector to rotate.

Return type numpy.ndarray

xiuminglib.geometry.rot.**is_rot_mat**(*mat*, *tol=1e-06*)
Checks if a matrix is a valid rotation matrix.

Parameters

- **mat** (*numpy.ndarray*) – A 3×3 matrix.
- **tol** (*float, optional*) – Tolerance for checking if all close.

Returns Whether this is a valid rotation matrix.

Return type bool

xiuminglib.geometry.rot.**rot_mat_to_euler_angles**(*rot_mat*, *tol=1e-06*)
Converts a rotation matrix into Euler angles (rotation angles around the x, y, and z axes).

Parameters

- **rot_mat** (*numpy.ndarray*) – 3×3 rotation matrix.
- **tol** (*float, optional*) – Tolerance for checking singularity.

Returns Euler angles in radians.

Return type numpy.ndarray

xiuminglib.geometry.sph module

xiuminglib.geometry.sph.**cart2sph**(*pts_cart*, *convention='lat-lng'*)
Converts 3D Cartesian coordinates to spherical coordinates.

Parameters

- **pts_cart** (*array_like*) – Cartesian x, y and z. Of shape N-by-3 or length 3 if just one point.

- **convention** (*str, optional*) – Convention for spherical coordinates: `'lat-lng'` or `'theta-phi'`:

```
lat-lng
                          ^ z (lat = 90)
                          |
                          |
    (lng = -90) ----------+----------> y (lng = 90)
                        ,'|
                      ,'  |
(lat = 0, lng = 0) x      | (lat = -90)
```

```
theta-phi
                             ^ z (theta = 0)
                             |
                             |
       (phi = 270) ----------+----------> y (phi = 90)
                           ,'|
                         ,'  |
(theta = 90, phi = 0) x      | (theta = 180)
```

Returns Spherical coordinates (r, θ_1, θ_2) in radians.

Return type numpy.ndarray

xiuminglib.geometry.sph.**main**(*func_name*)
Unit tests that can also serve as example usage.

xiuminglib.geometry.sph.**sph2cart**(*pts_sph*, *convention='lat-lng'*)
Inverse of *cart2sph()*.

See *cart2sph()*.

xiuminglib.geometry.sph.**uniform_sample_sph**(*n*, *r=1*, *convention='lat-lng'*)
Uniformly samples points on the sphere [source].

Parameters

- **n** (*int*) – Total number of points to sample. Must be a square number.
- **r** (*float, optional*) – Radius of the sphere. Defaults to 1.
- **convention** (*str, optional*) – Convention for spherical coordinates. See *cart2sph()* for conventions.

Returns Spherical coordinates (r, θ_1, θ_2) in radians. The points are ordered such that all azimuths are looped through first at each elevation.

Return type numpy.ndarray

xiuminglib.geometry.tri module

xiuminglib.geometry.tri.**barycentric**(*pts*, *tvs*)
Computes barycentric coordinates of 3D point(s) w.r.t. a triangle.

Parameters

- **pts** (*array_like*) – 3-array for one point; N-by-3 array for multiple points.
- **tvs** (*array_like*) – 3-by-3 array with rows being the triangle's vertices.

Returns Barycentric coordinates of the same shape as `pts`. If any array element $\notin [0, 1]$, the input point doesn't fall on the triangle.

Return type numpy.ndarray

xiuminglib.geometry.tri.**moeller_trumbore**(*ray_orig*, *ray_dir*, *tri_v0*, *tri_v1*, *tri_v2*)

Decides if a ray intersects with a triangle using the Moeller-Trumbore algorithm.

$O + D = (1 - u - v)V_0 + uV_1 + vV_2$.

Parameters

- **ray_orig** (*array_like*) – 3D coordinates of the ray origin O.
- **ray_dir** (*array_like*) – Ray direction D (not necessarily normalized).
- **tri_v0** (*array_like*) – Triangle vertex V_0.
- **tri_v1** (*array_like*) – Triangle vertex V_1.
- **tri_v2** (*array_like*) – Triangle vertex V_2.

Returns

- **u** (*float*) – The u component of the Barycentric coordinates of the intersection. Intersection is in-triangle (including on an edge or at a vertex), if $u \geq 0$, $v \geq 0$, and $u + v \leq 1$.
- **v** (*float*) – The v component.
- **t** (*float*) – Distance coefficient from O to the intersection along D. Intersection is between O and $O + D$, if $0 < t < 1$.

Return type tuple

xiuminglib.io package

Submodules

xiuminglib.io.exr module

xiuminglib.io.exr.**read**(*path*)

Reads a non-multi-layer OpenEXR image from disk.

Reading a multi-layer OpenEXR cannot be done with OpenCV and would require installing OpenEXR and Imath (see *cli/exr2npz.py*).

Parameters **path** (*str*) – Path to the .exr file.

Returns Loaded (float) array with RGB channels in order.

Return type numpy.ndarray

xiuminglib.io.hdr module

xiuminglib.io.hdr.**read**(*path*)

Reads an HDR map from disk.

Parameters **path** (*str*) – Path to the .hdr file.

Returns Loaded (float) HDR map with RGB channels in order.

Return type numpy.ndarray

xiuminglib.io.hdr.**write**(*rgb*, *outpath*)
Writes a `float32` array as an HDR map to disk.

Parameters

- **rgb** (*numpy.ndarray*) – `float32` RGB array.
- **outpath** (*str*) – Output path.

Writes

- The resultant HDR map.

xiuminglib.io.img module

xiuminglib.io.img.**load**(**args*, ***kwargs*)
Alias for *read()*, mostly for backward compatibility.

TODO: remove

xiuminglib.io.img.**read**(*path*, *auto_rotate=False*)
Reads an image from disk.

Parameters

- **path** (*str*) – Path to the image file. Supported formats: whatever Pillow supports.
- **auto_rotate** (*bool, optional*) – Whether to auto-rotate the read image array according to its EXIF orientation, if any.

Returns Loaded image.

Return type numpy.ndarray

xiuminglib.io.img.**write_arr**(**args*, ***kwargs*)
Alias for *write_float()*, mostly for backward compatibility.

TODO: remove

xiuminglib.io.img.**write_float**(*arr_0to1*, *outpath*, *img_dtype='uint8'*, *clip=False*)
Writes a `float` array as an image to disk.

Parameters

- **arr_0to1** (*numpy.ndarray*) – Array with values roughly $\in [0, 1]$.
- **outpath** (*str*) – Output path.
- **img_dtype** (*str, optional*) – Image data type. Defaults to `'uint8'`.
- **clip** (*bool, optional*) – Whether to clip values to $[0, 1]$. Defaults to `False`.

Writes

- The resultant image.

Returns The resultant image array.

Return type numpy.ndarray

xiuminglib.io.img.**write_img**(**args*, ***kwargs*)
Alias for *write_uint()*, mostly for backward compatibility.

TODO: remove

xiuminglib.io.img.**write_uint**(*arr_uint*, *outpath*)
Writes an `uint` array as an image to disk.

Parameters

- **arr_uint** (*numpy.ndarray*) – A `uint` array.
- **outpath** (*str*) – Output path.

Writes

- The resultant image.

xiuminglib.io.json module

xiuminglib.io.json.**load**(*path*)
Loads a JSON.

Parameters **path** (*str*) – Path to the JSON file.

Returns Data dictionary.

xiuminglib.io.json.**write**(*dict_*, *path*)
Writes a dictionary into a JSON.

Parameters

- **dict** (*dict*) – Data dictionary.
- **path** (*str*) – Path to the JSON file.

Writes

- JSON file.

xiuminglib.io.np module

xiuminglib.io.np.**read_or_write**(*data_f*, *fallback=None*)
Loads the data file if it exists. Otherwise, if fallback is provided, call fallback and save its return to disk.

Parameters

- **data_f** (*str*) – Path to the data file, whose extension will be used for deciding how to load the data.
- **fallback** (*function, optional*) – Fallback function used if data file doesn't exist. Its return will be saved to `data_f` for future loadings. It should not take arguments, but if yours requires taking arguments, just wrap yours with:

```
fallback=lambda: your_fancy_func(var0, var1)
```

Returns Data loaded if `data_f` exists; otherwise, `fallback`'s return (`None` if no fallback).

Writes

- Return by the fallback, if provided.

xiuminglib.io.objmtl module

class xiuminglib.io.objmtl.**Mtl**(*obj*, *Ns=96.078431*, *Ka=(1, 1, 1)*, *Kd=(0.64, 0.64, 0.64)*, *Ks=(0.5, 0.5, 0.5)*, *Ni=1*, *d=1*, *illum=2*)

Bases: object

Wavefront .mtl object.

mtlfile
Material file name, set to obj.mtllib.

Type str

newmtl
Material name, set to obj.usemtl.

Type str

map_Kd_path
Path to the diffuse map, set to obj.diffuse_map_path.

Type str

map_Kd_scale
Scale of the diffuse map, set to obj.diffuse_map_scale.

Type float

Ns

Type float

Ka

Type tuple

Kd

Type tuple

Ks

Type tuple

Ni

Type float

d

Type float

illum

Type int

__init__(*obj*, *Ns=96.078431*, *Ka=(1, 1, 1)*, *Kd=(0.64, 0.64, 0.64)*, *Ks=(0.5, 0.5, 0.5)*, *Ni=1*, *d=1*, *illum=2*)

Parameters

- **obj** (Obj) – Obj object for which this Mtl object is created.
- **Ns** (*float, optional*) – Specular exponent, normally $\in [0, 1000]$.
- **Ka** (*tuple, optional*) – Ambient reflectivity, each float normally $\in [0, 1]$. Values outside increase or decrease relectivity accordingly.
- **Kd** (*tuple, optional*) – Diffuse reflectivity. Same range as Ka.

- **Ks** (*tuple, optional*) – Specular reflectivity. Same range as `Ka`.
- **Ni** (*float, optional*) – Optical density, a.k.a. index of refraction $\in [0.001, 10]$. 1 means light doesn't bend as it passes through. Increasing it increases the amount of bending. Glass has an index of refraction of about 1.5. Values of less than 1.0 produce bizarre results and are not recommended.
- **d** (*float, optional*) – Amount this material dissolves into the background $\in [0, 1]$. 1.0 is fully opaque (default), and 0 is fully dissolved (completely transparent). Unlike a real transparent material, the dissolve does not depend upon material thickness, nor does it have any spectral character. Dissolve works on all illumination models.
- **illum** (*int, optional*) – Illumination model $\in [0, 1, ..., 10]$.

print_info()

write_file(*outdir*)

Unit tests that can also serve as example usage.

Parameters **outdir** (*str*) – Output directory.

Writes

- Output .mtl file.

class xiuminglib.io.objmtl.**Obj**(*o=None*, *v=None*, *f=None*, *vn=None*, *fn=None*, *vt=None*, *ft=None*, *s=False*, *mtllib=None*, *usemtl=None*, *diffuse_map_path=None*, *diffuse_map_scale=1*)

Bases: `object`

Wavefront .obj Object.

Face, vertex, or other indices here all start from 1.

o

Type str

v

Type numpy.ndarray

f

Type list

vn

Type numpy.ndarray

fn

Type list

vt

Type numpy.ndarray

ft

Type list

s

Type bool

mtllib

Type str

usemtl

Type str

diffuse_map_path

Type str

diffuse_map_scale

Type float

__init__(*o=None*, *v=None*, *f=None*, *vn=None*, *fn=None*, *vt=None*, *ft=None*, *s=False*, *mtllib=None*, *usemtl=None*, *diffuse_map_path=None*, *diffuse_map_scale=1*)

Parameters

- **o** (*str, optional*) – Object name.
- **v** (*numpy.ndarray, optional*) – Vertex coordinates.
- **f** (*list, optional*) – Faces' vertex indices (1-indexed), e.g., `[[1, 2, 3], [4, 5, 6], [7, 8, 9, 10], ...]`.
- **vn** (*numpy.ndarray, optional*) – Vertex normals of shape N-by-3, normalized or not.
- **fn** (*list, optional*) – Faces' vertex normal indices, e.g., `[[1, 1, 1], [], [2, 2, 2, 2], ...]`. Must be of the same length as `f`.
- **vt** (*numpy.ndarray, optional*) – Vertex texture coordinates of shape N-by-2. Coordinates must be normalized to $[0, 1]$.
- **ft** (*list, optional*) – Faces' texture vertex indices, e.g., `[[1, 2, 3], [4, 5, 6], [], ...]`. Must be of the same length as `f`.
- **s** (*bool, optional*) – Group smoothing.
- **mtllib** (*str, optional*) – Material file name, e.g., `'cube.mtl'`.
- **usemtl** (*str, optional*) – Material name (defined in .mtl file).
- **diffuse_map_path** (*str, optional*) – Path to diffuse texture map.
- **diffuse_map_scale** (*float, optional*) – Scale of diffuse texture map.

load_file(*obj_file*)

Loads a (basic) .obj file as an object.

Populates attributes with contents read from file.

Parameters **obj_file** (*str*) – Path to .obj file.

print_info()

set_face_normals()

Sets face normals according to geometric vertices and their orders in forming faces.

Returns

- **vn** (*numpy.ndarray*) – Normal vectors.
- **fn** (*list*) – Normal faces. Each member list consists of the same integer, e.g., `[[1, 1, 1], [2, 2, 2, 2], ...]`.

Return type tuple

write_file(*objpath*)
Writes the current model to a .obj file.

Parameters **objpath** (*str*) – Path to the output .obj.

Writes

- Output .obj file.

xiuminglib.io.objmtl.**main**()
Unit tests that can also serve as example usage.

xiuminglib.vis package

Submodules

xiuminglib.vis.anim module

xiuminglib.vis.anim.**make_anim**(*imgs*, *duration=1*, *outpath=None*)
Writes a list of images into an animation.

In most cases, we need to label each image, for which you can use `vis.text.put_text()`.

Parameters

- **imgs** (*list(numpy.ndarray or str)*) – An image is either a path or an array (mixing ok, but arrays will need to be written to a temporary directory). If array, should be of type `uint` and of shape H-by-W (grayscale) or H-by-W-by-3 (RGB).
- **duration** (*float, optional*) – Duration of each frame in seconds.
- **outpath** (*str, optional*) – Where to write the output to (a .apng or .gif file). `None` means `os.path.join(const.Dir.tmp, 'make_anim.gif')`.

Writes

- An animation of the images.

xiuminglib.vis.general module

This module should be imported before `skimage` to avoid the `matplotlib` backend problem.

xiuminglib.vis.general.**axes3d_wrapper**(**args*, *func='scatter'*, *labels=None*, *legend_fontsize=20*, *legend_loc=0*, *figsize=(14, 14)*, *figtitle=None*, *figtitle_fontsize=20*, *xlabel=None*, *xlabel_fontsize=20*, *ylabel=None*, *ylabel_fontsize=20*, *zlabel=None*, *zlabel_fontsize=20*, *xticks=None*, *xticks_fontsize=10*, *xticks_rotation=0*, *yticks=None*, *yticks_fontsize=10*, *yticks_rotation=0*, *zticks=None*, *zticks_fontsize=10*, *zticks_rotation=0*, *grid=True*, *views=None*, *equal_axes=False*, *outpath=None*, ***kwargs*)
Convinience wrapper for `mpl_toolkits.mplot3d.Axes3D` functions.

It saves plots directly to the disk without displaying.

Parameters

- ***args** – Positional parameters that the wrapped function takes. See `mpl_toolkits.mplot3d.Axes3D`.
- ****kwargs** – Keyword parameters.
- **func** (*str, optional*) – Which pyplot function to invoke, e.g., `'scatter'`.
- **labels** (*list(str), optional*) – Labels for plot objects, to appear in the legend. Use `None` for no label for a certain object. `None` means no legend at all.
- **legend_loc** (*str, optional*) – Legend location: `'best'`, `'upper right'`, `'lower left'`, `'right'`, `'center left'`, `'lower center'`, `'upper center'`, `'center'`, etc. Effective only when `labels` is not `None`.
- **figsize** (*tuple, optional*) – Width and height of the figure in inches.
- **figtitle** (*str, optional*) – Figure title.
- **xlabel** (*str, optional*) – Label of x-axis.
- **ylabel** –
- **zlabel** –
- **xticks** (*array_like, optional*) – Tick values of x-axis. `None` means auto.
- **yticks** –
- **zticks** –
- ***_fontsize** (*int, optional*) – Font size.
- ***_rotation** (*float, optional*) – Tick rotation in degrees.
- **grid** (*bool, optional*) – Whether to draw grid.
- **views** (*list(tuple), optional*) – List of elevation-azimuth angle pairs (in degrees). A good set of views is `[(30, 0), (30, 45), (30, 90), (30, 135)]`.
- **equal_axes** (*bool, optional*) – Whether to have the same scale for all axes.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. Should end with `'.png'` or `'.pkl'` (for offline interactive viewing). `None` means `os.path.join(const.Dir.tmp, 'axes3d_wrapper.png')`.

Writes

- One or multiple (if `views` is provided) views of the 3D plot.

xiuminglib.vis.general.**make_colormap**(*low*, *high*)

Generates your own colormap for heatmap.

Parameters

- **low** (*str or tuple*) – Color for the lowest value, such as `'red'` or `(1, 0, 0)`.
- **high** –

Returns Generated colormap.

Return type matplotlib.colors.LinearSegmentedColormap

xiuminglib.vis.general.**pyplot_wrapper**(**args*, *ci=None*, *func='plot'*, *labels=None*, *legend_fontsize=20*, *legend_loc=0*, *figsize=(14, 14)*, *figtitle=None*, *figtitle_fontsize=20*, *xlabel=None*, *xlabel_fontsize=20*, *ylabel=None*, *ylabel_fontsize=20*, *xticks=None*, *xticks_locations=None*, *xticks_fontsize=10*, *xticks_rotation=0*, *yticks=None*, *yticks_locations=None*, *yticks_fontsize=10*, *yticks_rotation=0*, *xlim=None*, *ylim=None*, *grid=True*, *outpath=None*, ***kwargs*)

Convinience wrapper for `matplotlib.pyplot` functions.

It saves plots directly to the disk without displaying.

Parameters

- ***args** – Positional parameters that the wrapped function takes. See `matplotlib.pyplot`.
- ****kwargs** – Keyword parameters.
- **ci** (*list(float) or list(list(float)), optional*) – Confidence interval for `x_i[j]` is `y_i[j] +/- ci[i][j]`. Effective only when `func` is `'plot'`. List of floats for one line, and list of lists of floats for multiple lines.
- **func** (*str, optional*) – Which `pyplot` function to invoke, e.g., `'plot'` or `'bar'`.
- **labels** (*list, optional*) – Labels for plot objects, to appear in the legend. `None` means no label for this object.
- **legend_loc** (*str, optional*) – Legend location: `'best'`, `'upper right'`, `'lower left'`, `'right'`, `'center left'`, `'lower center'`, `'upper center'`, `'center'`, etc. Effective only when `labels` is not `None`.
- **figsize** (*tuple, optional*) – Width and height of the figure in inches.
- **figtitle** (*str, optional*) – Figure title.
- **xlabel** (*str, optional*) – Label of x-axis.
- **ylabel** –
- **xticks** (*array_like, optional*) – Tick values of x-axis. `None` means auto.
- **yticks** –
- **xticks_locations** (*array_like, optional*) – Locations of the ticks. `None` means starting from 0 and one next to another.
- **yticks_locations** –
- ***_fontsize** (*int, optional*) – Font size.
- ***_rotation** (*float, optional*) – Tick rotation in degrees.
- **xlim** (*list, optional*) – Start and end values for x-axis. `None` means auto.
- **ylim** –
- **grid** (*bool, optional*) – Whether to draw grid.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. `None` means `os.path.join(const.Dir.tmp, 'pyplot_wrapper.png')`.

Writes

- The plot.

xiuminglib.vis.geometry module

xiuminglib.vis.geometry.**depth_as_image**(*depth_map*, *alpha_map=None*, *keep_alpha=False*, *outpath=None*)

Visualizes a(n) (aliased) depth map and an (anti-aliased) alpha map as a single depth image.

Output has black background, with bright values for closeness to the camera. If the alpha map is anti-aliased, the result depth map will be nicely anti-aliased.

Parameters

- **depth_map** (*numpy.ndarray*) – 2D array of (aliased) raw depth values.
- **alpha_map** (*numpy.ndarray, optional*) – 2D array of (anti-aliased) alpha values.
- **keep_alpha** (*bool, optional*) – Whether to keep alpha channel in output.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. `None` means `os.path.join(const.Dir.tmp, 'depth_as_image.png')`.

Writes

- The (anti-aliased) depth image.

xiuminglib.vis.geometry.**normal_as_image**(*normal_map*, *alpha_map=None*, *keep_alpha=False*, *outpath=None*)

Visualizes the normal map by converting vectors to pixel values.

If not keeping alpha, the background is black, complying with industry standards (e.g., Adobe AE).

Parameters

- **normal_map** (*numpy.ndarray*) – H-by-W-by-3 array of normal vectors.
- **alpha_map** (*numpy.ndarray, optional*) – H-by-W array of alpha values.
- **keep_alpha** (*bool, optional*) – Whether to keep alpha channel in output.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. `None` means `os.path.join(const.Dir.tmp, 'normal_as_image.png')`.

Writes

- The normal image.

xiuminglib.vis.geometry.**ptcld_as_isosurf**(*pts*, *out_obj*, *res=128*, *center=False*)

Visualizes point cloud as isosurface of its TDF.

Parameters

- **pts** (*array_like*) – Cartesian coordinates in object space, of shape N-by-3.
- **out_obj** (*str*) – The output path of the surface .obj.
- **res** (*int, optional*) – Resolution of the TDF.
- **center** (*bool, optional*) – Whether to center these points around object space origin.

Writes

- The .obj file of the isosurface.

xiuminglib.vis.html module

class xiuminglib.vis.html.**HTML**(*title='Results'*, *bgcolor='black'*, *text_font='roboto'*, *text_color='white'*)

Bases: object

HTML Builder.

head

Type str

body

Type str

tail

Concatenating the three gives the complete file contents, as a string.

Type str

children

Child elements, such as a table.

Type dict

__init__(*title='Results'*, *bgcolor='black'*, *text_font='roboto'*, *text_color='white'*)

Parameters

- **title** (*str, optional*) – Page title.
- **bgcolor** (*str, optional*) – Background color. Supports at least color names (like, `'black'`) and Hex colors (like `'#FFFFFF'`).
- **text_font** (*str, optional*) – Supported values include `'arial'`, etc.
- **text_color** (*str, optional*) – Text color.

add_header(*text*, *level=1*)

add_table(*name=None*, *header=None*, *width='100%'*, *border=6*)

Adds a table to the HTML's children.

Parameters

- **name** (*str, optional*) – Table name to enable easy access: `self.children[name]`.
- **header** (*list(str), optional*) – Table header.
- **width** (*str, optional*) – Table width.
- **border** (*int, optional*) – Border width.

Returns Table added.

Return type *xiuminglib.vis.html.Table*

save(*index_file*)

Saves the generated HTML string to the index file.

Once called, this method also calls all children's `close()`, so that everything (e.g., a table) is properly closed.

Parameters **index_file** (*str*) – Path to the generated index.html.

Writes

- An HTML index file.

class xiuminglib.vis.html.**Table**(*header=None*, *width='100%'*, *border=6*)

Bases: object

HTML Table.

head

Type str

body

Type str

tail

Concatenating the three gives the complete file contents, as a string.

Type str

td

`td` start string that specifies a uniform style.

Type str

__init__(*header=None*, *width='100%'*, *border=6*)

Parameters

- **header** (*list(str), optional*) – Table headers.
- **width** (*str, optional*) – Table width.
- **border** (*int, optional*) – Border width.

add_row(*media*, *types*, *captions=None*, *media_width=256*)

Adds a row to the table.

Parameters

- **media** (*list(str)*) – Paths to media, or the text to display itself.
- **types** (*list(str)*) – Types for all the media: `'image'` or `'text'`.
- **captions** (*list(str), optional*) – Media captions to appear below the media.
- **media_width** (*int, optional*) – Media width in pixels.

close()

Closes the table.

Returns The generated table as a string.

Return type str

xiuminglib.vis.matrix module

xiuminglib.vis.matrix.**matrix_as_heatmap**(*mat*, *cmap='viridis'*, *center_around_zero=False*, *outpath=None*, *contents_only=False*, *figtitle=None*)

Visualizes a matrix as heatmap.

Parameters

- **mat** (*numpy.ndarray*) – Matrix to visualize as heatmp. May contain NaN's, which will be plotted white.
- **cmap** (*str, optional*) – Colormap to use.
- **center_around_zero** (*bool, optional*) – Whether to center colorbar around 0 (so that zero is no color, i.e., white). Useful when matrix consists of both positive and negative values, and 0 means "nothing". `None` means default colormap and auto range.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. `None` means `os.path.join(const.Dir.tmp, 'matrix_as_heatmap.png')`.
- **contents_only** (*bool, optional*) – Whether to plot only the contents (i.e., no borders, axes, etc.). If `True`, the heatmap will be of exactly the same size as your matrix, useful when you want to plot heatmaps separately and later concatenate them into a single one.
- **figtitle** (*str, optional*) – Figure title. `None` means no title.

Writes

- A heatmap of the matrix.

xiuminglib.vis.matrix.**matrix_as_heatmap_complex**(**args*, ***kwargs*)

Wraps *matrix_as_heatmap()* for complex number support.

Just pass in the parameters that *matrix_as_heatmap()* takes. `'_mag'` and `'_phase'` will be appended to `outpath` to produce the magnitude and phase heatmaps, respectively. Specifically, magnitude is computed by `numpy.absolute()`, and phase by `numpy.angle()`.

Writes

- A magnitude heatmap with `'_mag'` in its filename.
- A phase heatmap with `'_phase'` in its filename.

xiuminglib.vis.plot module

class xiuminglib.vis.plot.**Plot**(*legend_fontsize=20*, *legend_loc=0*, *figsize=(14, 14)*, *figtitle=None*, *figtitle_fontsize=20*, *xlabel=None*, *xlabel_fontsize=20*, *ylabel=None*, *ylabel_fontsize=20*, *zlabel=None*, *zlabel_fontsize=20*, *xlim=None*, *ylim=None*, *zlim=None*, *xticks=None*, *xticks_fontsize=10*, *xticks_rotation=0*, *yticks=None*, *yticks_fontsize=10*, *yticks_rotation=0*, *zticks=None*, *zticks_fontsize=10*, *zticks_rotation=0*, *grid=True*, *labels=None*, *outpath=None*)

Bases: `object`

__init__(*legend_fontsize=20*, *legend_loc=0*, *figsize=(14, 14)*, *figtitle=None*, *figtitle_fontsize=20*, *xlabel=None*, *xlabel_fontsize=20*, *ylabel=None*, *ylabel_fontsize=20*, *zlabel=None*, *zlabel_fontsize=20*, *xlim=None*, *ylim=None*, *zlim=None*, *xticks=None*, *xticks_fontsize=10*, *xticks_rotation=0*, *yticks=None*, *yticks_fontsize=10*, *yticks_rotation=0*, *zticks=None*, *zticks_fontsize=10*, *zticks_rotation=0*, *grid=True*, *labels=None*, *outpath=None*)

Plotter.

Parameters

- **legend_fontsize** (*int, optional*) – Legend font size.

- **legend_loc** (*str, optional*) – Legend location: `'best'`, `'upper right'`, `'lower left'`, `'right'`, `'center left'`, `'lower center'`, `'upper center'`, `'center'`, etc. Effective only when `labels` is not `None`.
- **figsize** (*tuple, optional*) – Width and height of the figure in inches.
- **figtitle** (*str, optional*) – Figure title.
- ***_fontsize** (*int, optional*) – Font size.
- **?label** (*str, optional*) – Axis labels.
- **?lim** (*array_like, optional*) – Axis min. and max. `None` means auto.
- **?ticks** (*array_like, optional*) – Axis tick values. `None` means auto.
- **?ticks_rotation** (*float, optional*) – Tick rotation in degrees.
- **grid** (*bool, optional*) – Whether to draw grid.
- **labels** (*list, optional*) – Labels.
- **outpath** (*str, optional*) – Path to which the plot is saved to. Should end with `'.png'`, and `None` means to `const.Dir.tmp`.

bar(*y*, *group_width=0.8*)
Bar plot.

Parameters

- **y** (*array_like*) – N-by-M array of N groups, each with M bars, or N-array of N groups, each with one bar.
- **group_width** (*float, optional*) – Width allocated to each group, shared by all bars within the group.

Writes

- The bar plot.

line(*xy*, *width=None*, *marker=None*, *marker_size=None*)
Line/curve plot.

Parameters

- **xy** (*array_like*) – N-by-M array of N x-values (first column) and their corresponding y-values (the remaining M-1 columns).
- **width** (*float, optional*) – Line width.
- **marker** (*str, optional*) – Marker.
- **marker_size** (*float, optional*) – Marker size.

Writes

- The line plot.

scatter3d(*xyz*, *colors=None*, *size=None*, *equal_axes=False*, *views=None*)
3D scatter plot.

Parameters

- **xyz** (*array_like*) – N-by-3 array of N points.

- **colors** (*array_like or list(str) or str, optional*) – If N-array, these values are colormapped. If N-list, its elements should be color strings. If a single color string, all points use that color.
- **size** (*int, optional*) – Scatter size.
- **equal_axes** (*bool, optional*) – Whether to have the same scale for all axes.
- **views** (*list(tuple), optional*) – List of elevation-azimuth angle pairs (in degrees). A good set of views is `[(30, 0), (30, 45), (30, 90), (30, 135)]`.

Writes

- One or multiple (if `views` is provided) views of the 3D plot.

xiuminglib.vis.pt module

xiuminglib.vis.pt.**scatter_on_img**(*pts*, *im*, *size=2*, *bgr=(0, 0, 255)*, *outpath=None*)

Plots scatter on top of an image or just a white canvas, if you are being creative by feeding in just a white image.

Parameters

- **pts** (*array_like*) – Pixel coordinates of the scatter point(s), of length 2 for just one point or shape N-by-2 for multiple points. Convention:

```
+-----------> dim1
|
|
|
v dim0
```

- **im** (*numpy.ndarray*) – Image to scatter on. H-by-W (grayscale) or H-by-W-by-3 (RGB) arrays of `unint` type.
- **size** (*float or array_like(float), optional*) – Size(s) of scatter points. If *array_like*, must be of length N.
- **bgr** (*tuple or array_like(tuple), optional*) – BGR color(s) of scatter points. Each element $\in [0, 255]$. If *array_like*, must be of shape N-by-3.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. `None` means `os.path.join(const.Dir.tmp, 'scatter_on_img.png')`.

Writes

- The scatter plot overlaid over the image.

xiuminglib.vis.pt.**uv_on_texmap**(*uvs*, *texmap*, *ft=None*, *outpath=None*, *max_n_lines=None*, *dotsize=4*, *dotcolor='r'*, *linewidth=1*, *linecolor='b'*)

Visualizes which points on texture map the vertices map to.

Parameters

- **uvs** (*numpy.ndarray*) – N-by-2 array of UV coordinates. See *xiuminglib.blender.object.smart_uv_unwrap()* for the UV coordinate convention.
- **texmap** (*numpy.ndarray or str*) – Loaded texture map or its path. If *numpy.ndarray*, can be H-by-W (grayscale) or H-by-W-by-3 (color).

- **ft** (*list(list(int)), optional*) – Texture faces used to connect the UV points. Values start from 1, e.g., `'[[1, 2, 3], [], [2, 3, 4, 5], ...]'`.
- **outpath** (*str, optional*) – Path to which the visualization is saved to. `None` means `os.path.join(const.Dir.tmp, 'uv_on_texmap.png')`.
- **max_n_lines** (*int, optional*) – Plotting a huge number of lines can be slow, so set this to uniformly sample a subset to plot. Useless if `ft` is `None`.
- **dotsize** (*int or list(int), optional*) – Size(s) of the UV dots.
- **dotcolor** (*str or list(str), optional*) – Their color(s).
- **linewidth** (*float, optional*) – Width of the lines connecting the dots.
- **linecolor** (*str, optional*) – Their color.

Writes

- An image of where the vertices map to on the texture map.

xiuminglib.vis.text module

xiuminglib.vis.text.**put_text**(*img*, *text*, *label_top_left_xy=None*, *font_size=None*, *font_color=(1, 0, 0)*, *font_ttf=None*)

Puts text on image.

Parameters

- **img** (*numpy.ndarray*) – Should be of type `uint` and of shape H-by-W (grayscale) or H-by-W-by-3 (RGB).
- **text** (*str*) – Text to be written on the image.
- **label_top_left_xy** (*tuple(int), optional*) – The XY coordinate of the label's top left corner.
- **font_size** (*int, optional*) – Font size.
- **font_color** (*tuple(float), optional*) – Font RGB, normalized to $[0, 1]$. Defaults to red.
- **font_ttf** (*str, optional*) – Path to the .ttf font file. Defaults to Arial.

Returns The modified image with text.

Return type numpy.ndarray

xiuminglib.vis.text.**text_as_image**(*text*, *imsize=256*, *thickness=2*, *dtype='uint8'*, *outpath=None*, *quiet=False*)

Rasterizes a text string into an image.

The text will be drawn in white to the center of a black canvas. Text size gets automatically figured out based on the provided thickness and image size.

Parameters

- **text** (*str*) – Text to be drawn.
- **imsize** (*float or tuple(float), optional*) – Output image height and width.
- **thickness** (*float, optional*) – Text thickness.
- **dtype** (*str, optional*) – Image type.

- **outpath** (*str, optional*) – Where to dump the result to. `None` means returning instead of writing it.
- **quiet** (*bool, optional*) – Whether to refrain from logging. Effective only when `outpath` is not `None`.

Returns or Writes

- An image of the text.

xiuminglib.vis.video module

xiuminglib.vis.video.**make_comparison_video**(*imgs1*, *imgs2*, *bar_width=4*, *bar_color=(1, 0, 0)*, *sweep_vertically=False*, *sweeps=1*, *label1=''*, *label2=''*, *font_size=None*, *font_ttf=None*, *label1_top_left_xy=None*, *label2_top_left_xy=None*, ***make_video_kwargs*)

Writes two lists of images into a comparison video that toggles between two videos with a sweeping bar.

Parameters

- **imgs?** (*list(numpy.ndarray)*) – Each image should be of type `uint8` or `uint16` and of shape H-by-W (grayscale) or H-by-W-by-3 (RGB).
- **bar_width** (*int, optional*) – Width of the sweeping bar.
- **bar_color** (*tuple(float), optional*) – Bar and label RGB, normalized to $[0, 1]$. Defaults to red.
- **sweep_vertically** (*bool, optional*) – Whether to sweep vertically or horizontally.
- **sweeps** (*int, optional*) – Number of sweeps.
- **label?** (*str, optional*) – Label for each video.
- **font_size** (*int, optional*) – Font size.
- **font_ttf** (*str, optional*) – Path to the .ttf font file. Defaults to Arial.
- **label?_top_left_xy** (*tuple(int), optional*) – The XY coordinate of the label's top left corner.
- **make_video_kwargs** (*dict, optional*) – Keyword arguments for *make_video()*.

Writes

- A comparison video.

xiuminglib.vis.video.**make_video**(*imgs*, *fps=24*, *outpath=None*, *method='matplotlib'*, *dpi=96*, *bitrate=-1*)

Writes a list of images into a grayscale or color video.

Parameters

- **imgs** (*list(numpy.ndarray)*) – Each image should be of type `uint8` or `uint16` and of shape H-by-W (grayscale) or H-by-W-by-3 (RGB).
- **fps** (*int, optional*) – Frame rate.

- **outpath** (*str, optional*) – Where to write the video to (a .mp4 file). `None` means `os.path.join(const.Dir.tmp, 'make_video.mp4')`.
- **method** (*str, optional*) – Method to use: `'matplotlib'`, `'opencv'`, `'video_api'`.
- **dpi** (*int, optional*) – Dots per inch when using `matplotlib`.
- **bitrate** (*int, optional*) – Bit rate in kilobits per second when using `matplotlib`; reasonable values include 7200.

Writes

- A video of the images.

3.1.2 Submodules

3.1.3 xiuminglib.camera module

class xiuminglib.camera.**PerspCam**(*name='cam', f_pix=533.33, im_res=(256, 256), loc=(1, 1, 1), lookat=(0, 0, 0), up=(0, 1, 0)*)

Bases: `object`

Perspective camera in 35mm format.

This is not an OpenGL/Blender camera (where $+x$ points right, $+y$ up, and $-z$ into the viewing direction), but rather a "CV camera" (where $+x$ points right, $+y$ down, and $+z$ into the viewing direction). See more in *ext_mat*.

Because we mostly consider just the camera and the object, we assume the object coordinate system (the "local system" in Blender) aligns with (and hence, is the same as) the world coordinate system (the "global system" in Blender).

Note:

- Sensor width of the 35mm format is actually 36mm.
- This class assumes unit pixel aspect ratio (i.e., $f_x = f_y$) and no skewing between the sensor plane and optical axis.
- The active sensor size may be smaller than `sensor_w` and `sensor_h`, depending on `im_res`. See *sensor_w_active* and *sensor_h_active*.
- `aov`, `sensor_h`, and `sensor_w` are hardware properties, having nothing to do with `im_res`.

__init__(*name='cam', f_pix=533.33, im_res=(256, 256), loc=(1, 1, 1), lookat=(0, 0, 0), up=(0, 1, 0)*)

Parameters

- **name** (*str, optional*) – Camera name.
- **f_pix** (*float, optional*) – Focal length in pixel.
- **im_res** (*array_like, optional*) – Image height and width in pixels.
- **loc** (*array_like, optional*) – Camera location in object space.
- **lookat** (*array_like, optional*) – Where the camera points to in object space, so default $(0, 0, 0)$ is the object center.

- **up** (*array_like, optional*) – Vector in object space that, when projected, points upward in image.

aov
Vertical and horizontal angles of view in degrees.

Type numpy.ndarray

backproj(*depth*, *fg_mask=None*, *bg_fill=0.0*, *depth_type='plane'*, *space='object'*)
Backprojects a depth map to 3D points.

Resolution of the depth map may be different from *im_h* and *im_w*: *im_h* and *im_w* decide the image coordinate bounds, and the depth resolution decides number of steps.

Parameters

- **depth** (*numpy.ndarray*) – Depth map.
- **fg_mask** (*numpy.ndarray, optional*) – Backproject only pixels falling inside this foreground mask. Its values should be logical.
- **bg_fill** (*flaot, optional*) – Filler value for background region.
- **depth_type** (*str, optional*) – Plane or ray depth.
- **space** (*str, optional*) – In which space the backprojected points are specified: `'object'` or `'camera'`.

Returns xyz map.

Return type numpy.ndarray

blender_rot_euler
Euler rotations in degrees.

Type numpy.ndarray

ext_mat
3×4 object-to-camera extrinsics matrix, i.e., rotation and translation that transform a point from object space to camera space.

Two coordinate systems involved: object space "obj" and camera space following the computer vision convention "cv", where $+x$ horizontally points right (to align with pixel coordinates), $+y$ vertically points down, and $+z$ is the look-at direction (because right-handed).

Type numpy.ndarray

ext_mat_4x4
Padding $[0, 0, 0, 1]$ to bottom of the 3×4 extrinsics matrix to make it invertible.

Type numpy.ndarray

f_mm
35mm format-equivalent focal length in mm.

Type float

f_pix
Focal length in pixels.

Type float

gen_rays(*spp=1*)
Generates ray directions in object space, with the ray origin being the camera location.

Parameters **spp** (*int, optional*) – Samples (or number of rays) per pixel. Must be a perfect square S^2 due to uniform, deterministic supersampling.

Returns An $H \times W \times S^2 \times 3$ array of ray directions.

Return type numpy.ndarray

get_cam2obj(*cam_type='cv'*, *square=False*)
Inverse of *get_obj2cam()*.

One example use: calling this with `cam_type='blender'` gives Blender's `cam.matrix_world`.

get_obj2cam(*cam_type='cv'*, *square=False*)
Gets the object-to-camera transformation matrix.

Parameters

- **cam_type** (*str, optional*) – Accepted are `'cv'`/`'opencv'` and `'opengl'`/`'blender'`.
- **square** (*bool, optional*) – If true, the last row of $[0, 0, 0, 1]$ is kept, which makes the matrix invertible.

Returns 3×4 or 4×4 object-to-camera transformation matrix.

Return type numpy.ndarray

im_h
Image height.

Type int

im_w
Image width.

Type int

int_mat
3×3 intrinsics matrix.

Type numpy.ndarray

loc
Camera location in object space.

Type numpy.ndarray

lookat
Where in object space the camera points to.

Type numpy.ndarray

mm_per_pix
Millimeter per pixel.

Type float

name
Camera name.

Type str

proj(*pts*, *space='object'*)
Projects 3D points to 2D.

Parameters

- **pts** (*array_like*) – 3D point(s) of shape $N \times 3$ or $3 \times N$, or of length 3.
- **space** (*str, optional*) – In which space these points are specified: `'object'` or `'camera'`.

Returns

Vertical and horizontal coordinates of the projections, following:

```
+-----------> dim1
|
|
|
v dim0
```

Return type array_like

proj_mat
3×4 projection matrix, derived from intrinsics and extrinsics.

Type numpy.ndarray

resize(*new_h=None*, *new_w=None*)
Updates the camera intrinsics according to the new size.

Parameters

- **new_h** (*int, optional*) – Target height. If `None`, will be calculated according to the target width, assuming the same aspect ratio.
- **new_w** (*int, optional*) – Target width. If `None`, will be calculated according to the target height, assuming the same aspect ratio.

sensor_fit_horizontal
Whether field of view angle fits along the horizontal or vertical direction.

Type bool

sensor_h
Sensor's physical height (fixed at 24mm).

Type float

sensor_h_active
Actual sensor height (mm) in use (resolution-dependent).

Type float

sensor_w
Sensor's physical width (fixed at 36mm).

Type float

sensor_w_active
Actual sensor width (mm) in use (resolution-dependent).

Type float

set_from_mitsuba(*xml_path*)
Sets camera according to a Mitsuba XML file.

Parameters **xml_path** (*str*) – Path to the XML file.

to_dict(*app=None*)
Converts this camera to a dictionary of its properties.

Parameters **app** (*str, optional*) – For what application are we converting? Accepted are `None` and `'blender'`.

Returns This camera as a dictionary.

Return type dict

up
Up vector, the vector in object space that, when projected, points upward on image plane.

Type numpy.ndarray

xiuminglib.camera.**safe_cast_to_int**(*x*)
Casts a string or float to integer only when safe.

Parameters **x** (*str or float*) – Input to be cast to integer.

Returns Integer version of the input.

Return type int

3.1.4 xiuminglib.const module

class xiuminglib.const.**Dir**
Bases: object

data = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stable/da

mstatus = '/tmp/machine-status/runtime'

tmp = '/tmp/'

class xiuminglib.const.**Path**
Bases: object

armadillo = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/sta

buddha = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stable

buddha_prefix = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts

buddha_res2 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/s

buddha_res3 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/s

buddha_res4 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/s

bunny = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stable/

bunny_prefix = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/

bunny_res2 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/st

bunny_res3 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/st

bunny_res4 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/st

cameraman = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/sta

checker = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stabl

cpustatus = '/tmp/cpu/machine_status.txt'

dragon = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stable

dragon_prefix = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts

dragon_res2 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/s

```
dragon_res3 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/s
dragon_res4 = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/s
gpustatus = '/tmp/gpu/{machine_name}'
lenna = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stable/
lpips_weights = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts
open_sans_regular = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/check
teapot = '/home/docs/checkouts/readthedocs.org/user_builds/xiuminglib/checkouts/stable
```

3.1.5 xiuminglib.decor module

Decorators that wrap a function.

If the function is defined in the file where you want to use the decorator, you can decorate the function at define time:

```
@decorator
def somefunc():
    return
```

If the function is defined somewhere else, do:

```
from numpy import mean

mean = decorator(mean)
```

xiuminglib.decor.**colossus_interface**(*somefunc*)

Wraps black-box functions to read from and write to Google Colossus.

Because it's hard (if possible at all) to figure out which path is input, and which is output, when the input function is black-box, this is a "best-effort" decorator (see below for warnings).

This decorator works by looping through all the positional and keyword parameters, copying CNS paths that exist prior to `somefunc` execution to temporary local locations, running `somefunc` and writing its output to local locations, and finally copying local paths that get modified by `somefunc` to their corresponding CNS locations.

> **Warning:** Therefore, if `somefunc`'s output already exists (e.g., you are re-running the function to overwrite the old result), it will be copied to local, overwritten by `somefunc` locally, and finally copied back to CNS. This doesn't lead to wrong behaviors, but is inefficient.

This decorator doesn't depend on Blaze, as it's using the `fileutil` CLI, rather than `google3.pyglib.gfile`. This is convenient in at least two cases:

- You are too lazy to use Blaze, want to run tests quickly on your local machine, but need access to CNS files.
- Your IO is more complex than what `with gfile.Open(...) as h:` can do (e.g., a Blender function importing an object from a path), in which case you have to copy the CNS file to local ("local" here could also mean a Borglet's local).

This interface generally works with resolved paths (e.g., `/path/to/file`), but not with wildcard paths (e.g., `/path/to/???`), sicne it's hard (if possible at all) to guess what your function tries to do with such wildcard paths.

Writes

- Input files copied from Colossus to `$TMP/`.
- Output files generated to `$TMP/`, to be copied to Colossus.

`xiuminglib.decor.`**`existok`**(*makedirs_func*)
Implements the `exist_ok` flag in 3.2+, which avoids race conditions, where one parallel worker checks the folder doesn't exist and wants to create it with another worker doing so faster.

`xiuminglib.decor.`**`main`**()
Unit tests that can also serve as example usage.

`xiuminglib.decor.`**`timeit`**(*somefunc*)
Outputs the time a function takes to execute.

3.1.6 xiuminglib.img module

`xiuminglib.img.`**`alpha_blend`**(*arr1*, *alpha*, *arr2=None*)
Alpha-blends two arrays, or masks one array.

Parameters

- **`arr1`** (*`numpy.ndarray`*) – Input array.
- **`alpha`** (*`numpy.ndarray`*) – Alpha map whose values are $\in [0, 1]$.
- **`arr2`** (*`numpy.ndarray`*) – Input array. If `None`, `arr1` will be blended with an all-zero array, equivalent to masking `arr1`.

Returns Blended array of type `float`.

Return type numpy.ndarray

`xiuminglib.img.`**`binarize`**(*im*, *threshold=None*)
Binarizes images.

Parameters

- **`im`** (*`numpy.ndarray`*) – Image to binarize. Of any integer type (`uint8`, `uint16`, etc.). If H-by-W-by-3, will be converted to grayscale and treated as H-by-W.
- **`threshold`** (*`float, optional`*) – Threshold for binarization. `None` means midpoint of the `dtype`.

Returns Binarized image. Of only 0's and 1's.

Return type numpy.ndarray

`xiuminglib.img.`**`compute_gradients`**(*im*)
Computes magnitudes and orientations of image gradients.

With Scharr operators:

```
[ 3 0 -3 ]           [ 3  10  3]
[10 0 -10]    and    [ 0   0  0]
[ 3 0 -3 ]           [-3 -10 -3]
```

Parameters **`im`** (*`numpy.ndarray`*) – H-by-W if single-channel (e.g., grayscale) or H-by-W-by-C if multi-channel (e.g., RGB) images. Gradients are computed independently for each of the C channels.

Returns

- **grad_mag** (*numpy.ndarray*) – Magnitude image of the gradients.
- **grad_orient** (*numpy.ndarray*) – Orientation image of the gradients (in radians).

```
       y ^ pi/2
         |
pi       |
 --------+--------> 0
-pi      |       x
         | -pi/2
```

Return type tuple

xiuminglib.img.**denormalize_float**(*arr*, *uint_type='uint8'*)

De-normalizes the input `float` array such that 1 becomes the target `uint` maximum.

Parameters

- **arr** (*numpy.ndarray*) – Input array of type `float`.
- **uint_type** (*str, optional*) – Target `uint` type.

Returns De-normalized array of the target type.

Return type numpy.ndarray

xiuminglib.img.**find_local_extrema**(*im*, *want_maxima*, *kernel_size=3*)

Finds local maxima or minima in an image.

Parameters

- **im** (*numpy.ndarray*) – H-by-W if single-channel (e.g., grayscale) or H-by-W-by-C for multi-channel (e.g., RGB) images. Extrema are found independently for each of the C channels.
- **want_maxima** (*bool*) – Whether maxima or minima are wanted.
- **kernel_size** (*int, optional*) – Side length of the square window under consideration. Must be larger than 1.

Returns Binary map indicating if each pixel is a local extremum.

Return type numpy.ndarray

xiuminglib.img.**gamma_correct**(*im*, *gamma=2.2*)

Applies gamma correction to an `uint` image.

Parameters

- **im** (*numpy.ndarray*) – H-by-W if single-channel (e.g., grayscale) or H-by-W-by-C multi-channel (e.g., RGB) `uint` images.
- **gamma** (*float, optional*) – Gamma value < 1 shifts image towards the darker end of the spectrum, while value > 1 towards the brighter.

Returns Gamma-corrected image.

Return type numpy.ndarray

xiuminglib.img.**grid_query_img**(*im*, *query_x*, *query_y*, *method='bilinear'*)

Grid queries an image via interpolation.

If you want to grid query unstructured data, consider *grid_query_unstruct()*.

This function uses either bilinear interpolation that allows you to break big matrices into patches and work locally, or bivariate spline interpolation that fits a global spline (so memory-intensive) and shows global effects.

Parameters

- **im** (*numpy.ndarray*) – H-by-W or H-by-W-by-C rectangular grid of data. Each of C channels is interpolated independently.
- **query_x** (*array_like*) – x coordinates of the queried rectangle, e.g., `np.arange(10)` for a 10-by-10 grid (hence, this should *not* be generated by `numpy.meshgrid()` or similar functions).
- **query_y** (*array_like*) – y coordinates, following this convention:

```
+---------> query_x
|
|
|
v query_y
```

- **method** (*str, optional*) – Interpolation method: `'spline'` or `'bilinear'`.

Returns Interpolated values at query locations, of shape `(len(query_y), len(query_x))` for single-channel input or `(len(query_y), len(query_x), im.shape[2])` for multi-channel input.

Return type numpy.ndarray

xiuminglib.img.**grid_query_unstruct**(*uvs*, *values*, *grid_res*, *method=None*)

Grid queries unstructured data given by coordinates and their values.

If you are looking to grid query structured data, such as an image, check out *grid_query_img()*.

This function interpolates values on a rectangular grid given some sparse, unstrucured samples. One use case is where you have some UV locations and their associated colors, and you want to "paint the colors" on a UV canvas.

Parameters

- **uvs** (*numpy.ndarray*) – N-by-2 array of UV coordinates where we have values (e.g., colors). See *xiuminglib.blender.object.smart_uv_unwrap()* for the UV coordinate convention.
- **values** (*numpy.ndarray*) – N-by-M array of M-D values at the N UV locations, or N-array of scalar values at the N UV locations. Channels are interpolated independently.
- **grid_res** (*array_like*) – Resolution (height first; then width) of the query grid.
- **method** (*dict, optional*) – Dictionary of method-specific parameters. Implemented methods and their default parameters:

```
# Default
method = {
    'func': 'griddata',
    # Which SciPy function to call.

    'func_underlying': 'linear',
    # Fed to `griddata` as the `method` parameter.

    'fill_value': (0,), # black
    # Will be used to fill in pixels outside the convex hulls
    # formed by the UV locations, and if `max_ll_interp` is
    # provided, also the pixels whose interpolation is too much
    # of a stretch to be trusted. In the context of "canvas
    # painting," this will be the canvas' base color.
```

(continues on next page)

(continued from previous page)

```
    'max_l1_interp': np.inf, # trust/accept all interpolations
    # Maximum L1 distance, which we can trust in interpolation,
    # to pixels that have values. Interpolation across a longer
    # range will not be trusted, and hence will be filled with
    # `fill_value`.
}
```

```
method = {
    'func': 'rbf',
    # Which SciPy function to call.

    'func_underlying': 'linear',
    # Fed to `Rbf` as the `method` parameter.

    'smooth': 0, # no smoothing
    # Fed to `Rbf` as the `smooth` parameter.
}
```

Returns Interpolated values at query locations, of shape `grid_res` for single-channel input or `(grid_res[0], grid_res[1], values.shape[2])` for multi-channel input.

Return type numpy.ndarray

`xiuminglib.img.`**`linear2srgb`**(*im*, *clip=False*)
Converts an image from linear RGB values to sRGB.

Parameters

- **`im`** (*`numpy.ndarray`*) – Of type `float`, and all pixels must be $\in [0, 1]$.
- **`clip`** (*`bool, optional`*) – Whether to clip values to $[0, 1]$. Defaults to `False`.

Returns Converted image in sRGB.

Return type numpy.ndarray

`xiuminglib.img.`**`normalize_uint`**(*arr*)
Normalizes the input `uint` array such that its `dtype` maximum becomes 1.

Parameters **`arr`** (*`numpy.ndarray`*) – Input array of type `uint`.

Returns Normalized array of type `float`.

Return type numpy.ndarray

`xiuminglib.img.`**`remove_islands`**(*im*, *min_n_pixels*, *connectivity=4*)
Removes small islands of pixels from a binary image.

Parameters

- **`im`** (*`numpy.ndarray`*) – Input binary image. Of only 0's and 1's.
- **`min_n_pixels`** (*`int`*) – Minimum island size to keep.
- **`connectivity`** (*`int, optional`*) – Definition of "connected": either 4 or 8.

Returns Output image with small islands removed.

Return type numpy.ndarray

`xiuminglib.img.`**`resize`**(*arr*, *new_h=None*, *new_w=None*, *method='cv2'*)
Resizes an image, with the option of maintaining the aspect ratio.

Parameters

- **arr** (*numpy.ndarray*) – Image to binarize. If multiple-channel, each channel is resized independently.
- **new_h** (*int, optional*) – Target height. If `None`, will be calculated according to the target width, assuming the same aspect ratio.
- **new_w** (*int, optional*) – Target width. If `None`, will be calculated according to the target height, assuming the same aspect ratio.
- **method** (*str, optional*) – Accepted values: `'cv2'` and `'tf'`.

Returns Resized image.

Return type numpy.ndarray

`xiuminglib.img.`**`rgb2lum`**(*im*)
Converts RGB to relative luminance (if input is linear RGB) or luma (if input is gamma-corrected RGB).

Parameters **im** (*numpy.ndarray*) – RGB array of shape `(..., 3)`.

Returns Relative luminance or luma array.

Return type numpy.ndarray

`xiuminglib.img.`**`srgb2linear`**(*im*, *clip=False*)
Converts an image from sRGB values to linear RGB.

Parameters

- **im** (*numpy.ndarray*) – Of type `float`, and all pixels must be $\in [0, 1]$.
- **clip** (*bool, optional*) – Whether to clip values to $[0, 1]$. Defaults to `False`.

Returns Converted image in linear RGB.

Return type numpy.ndarray

`xiuminglib.img.`**`tonemap`**(*hdr*, *method='gamma'*, *gamma=2.2*)
Tonemaps an HDR image.

Parameters

- **hdr** (*numpy.ndarray*) – HDR image.
- **method** (*str, optional*) – Values accepted: `'gamma'` and `'reinhard'`.
- **gamma** (*float, optional*) – Gamma value used if method is `'gamma'`.

Returns Tonemapped image $\in [0, 1]$.

Return type numpy.ndarray

3.1.7 xiuminglib.imprt module

`xiuminglib.imprt.`**`import_module_404ok`**(**args*, ***kwargs*)
Returns `None` (instead of failing) in the case of `ModuleNotFoundError`.

`xiuminglib.imprt.`**`preset_import`**(*name*, *assert_success=False*)
A unified importer for both regular and `google3` modules, according to specified presets/profiles (e.g., ignoring `ModuleNotFoundError`).

3.1.8 xiuminglib.interact module

xiuminglib.interact.**ask_to_proceed**(*msg*, *level='warning'*)
Pauses there to ask the user whether to proceed.

Parameters

- **msg** (*str*) – Message to display to the user.
- **level** (*str, optional*) – Message level, essentially deciding the message color: `'info'`, `'warning'`, or `'error'`.

xiuminglib.interact.**format_print**(*msg*, *fmt*)
Prints a message with format.

Parameters

- **msg** (*str*) – Message to print.
- **fmt** (*str*) – Format; try your luck with any value – don't worry; if it's illegal, you will be prompted with all legal values.

xiuminglib.interact.**print_attrs**(*obj*, *excerpts=None*, *excerpt_win_size=60*, *max_recursion_depth=None*)
Prints all attributes, recursively, of an object.

Parameters

- **obj** (*object*) – Object in which we search for the attribute.
- **excerpts** (*str or list(str), optional*) – Print only excerpts containing certain attributes. `None` means to print all.
- **excerpt_win_size** (*int, optional*) – How many characters get printed around a match.
- **max_recursion_depth** (*int, optional*) – Maximum recursion depth. `None` means no limit.

3.1.9 xiuminglib.linalg module

xiuminglib.linalg.**angle_between**(*vec1*, *vec2*, *radian=True*)
Computes the angle between two vectors.

Parameters

- **vec1** (*array_like*) – Vector 1.
- **vec2** –
- **radian** (*bool, optional*) – Whether to use radians.

Returns The angle $\in [0, \pi]$.

Return type float

xiuminglib.linalg.**calc_refl_vec**(*h*, *l*)
Calculates the reflection vector given the half vector.

Parameters

- **h** (*array_like*) – Half vector as a 3-array.
- **l** (*array_like*) – "Incident" vector (pointing outwards from the surface point), as a 3-array.

Returns Reflection vector as a 3-array.

Return type numpy.ndarray

xiuminglib.linalg.**is_identity**(*mat*, *eps=None*)
Checks if a matrix is an identity matrix.

If the input is not even square, `False` is returned.

Parameters

- **mat** (*numpy.ndarray*) – Input matrix.
- **eps** (*float, optional*) – Numerical tolerance for equality. `None` means `np.finfo(mat.dtype).eps`.

Returns Whether the input is an identity matrix.

Return type bool

xiuminglib.linalg.**is_symmetric**(*mat*, *eps=None*)
Checks if a matrix is symmetric.

If the input is not even square, `False` is returned.

Parameters

- **mat** (*numpy.ndarray*) – Input matrix.
- **eps** (*float, optional*) – Numerical tolerance for equality. `None` means `np.finfo(mat.dtype).eps`.

Returns Whether the input is symmetric.

Return type bool

xiuminglib.linalg.**main**(*func_name*)
Unit tests that can also serve as example usage.

xiuminglib.linalg.**normalize**(*vecs*, *axis=0*)
Normalizes vectors.

Parameters

- **vecs** (*array_like*) – 1D array for a single vector, 2D array for multiple vectors, 3D array for an "image" of vectors, etc.
- **axis** (*int, optional*) – Along which axis normalization is done.

Returns Normalized vector(s) of the same shape as input.

Return type numpy.ndarray

xiuminglib.linalg.**project_onto**(*pts*, *basis*)
Projects points onto a basis vector.

Parameters

- **pts** (*array_like*) – 1D array for one vector; 2D N-by-M array for N M-D points.
- **basis** (*array_like*) – 1D M-array specifying which basis vector to project to.

Returns Projected point(s) of the same shape.

Return type numpy.ndarray

xiuminglib.linalg.**solve_quadratic_eqn**(*a*, *b*, *c*)
Solves $ax^2 + bx + c = 0$.

3.1.10 xiuminglib.log module

xiuminglib.log.**get_logger**(*level=None*)
Creates a logger for functions in the library.

Parameters **level** (*str, optional*) – Logging level. Defaults to logging.INFO.

Returns Logger created.

Return type logging.Logger

3.1.11 xiuminglib.metric module

class xiuminglib.metric.**Base**(*dtype*)
Bases: object

The base metric.

dtype
Data type, with which data dynamic range is derived.

Type numpy.dtype

drange
Dynamic range, i.e., difference between the maximum and minimum allowed.

Type float

__call__(*im1*, *im2*, ***kwargs*)

Parameters

- **im1** (*numpy.ndarray*) – An image of shape H-by-W, H-by-W-by-1, or H-by-W-by-3.
- **im2** –

Returns The metric computed.

Return type float

__init__(*dtype*)

Parameters **dtype** (*str or numpy.dtype*) – Data type, from which dynamic range will be derived.

class xiuminglib.metric.**LPIPS**(*dtype*, *weight_pb=None*)
Bases: *xiuminglib.metric.Base*

The Learned Perceptual Image Patch Similarity (LPIPS) metric (lower is better).

Project page: https://richzhang.github.io/PerceptualSimilarity/

Note: This implementation assumes the minimum value allowed is 0, so data dynamic range becomes the maximum value allowed.

---

dtype
Data type, with which data dynamic range is derived.

Type numpy.dtype

drange
Dynamic range, i.e., difference between the maximum and minimum allowed.

Type float

lpips_func
The LPIPS network packed into a function.

Type tf.function

__call__(*im1*, *im2*)

Parameters

- **im1** –
- **im2** –

Returns LPIPS computed (lower is better).

Return type float

__init__(*dtype*, *weight_pb=None*)

Parameters

- **dtype** (*str or numpy.dtype*) – Data type, from which maximum allowed will be derived.
- **weight_pb** (*str, optional*) – Path to the network weight protobuf. Defaults to the bundled `net-lin_alex_v0.1.pb`.

class xiuminglib.metric.**PSNR**(*dtype*)
Bases: *xiuminglib.metric.Base*

Peak Signal-to-Noise Ratio (PSNR) in dB (higher is better).

If the inputs are RGB, they are first converted to luma (or relative luminance, if the inputs are not gamma-corrected). PSNR is computed on the luma.

__call__(*im1*, *im2*, *mask=None*)

Parameters

- **im1** –
- **im2** –
- **mask** (*numpy.ndarray, optional*) – An H-by-W logical array indicating pixels that contribute to the computation.

Returns PSNR in dB.

Return type float

class xiuminglib.metric.**SSIM**(*dtype*)
Bases: *xiuminglib.metric.Base*

The (multi-scale) Structural Similarity Index (SSIM) $\in [0, 1]$ (higher is better).

If the inputs are RGB, they are first converted to luma (or relative luminance, if the inputs are not gamma-corrected). SSIM is computed on the luma.

__call__(*im1*, *im2*, *multiscale=False*)

Parameters

- **im1** –
- **im2** –
- **multiscale** (*bool, optional*) – Whether to compute MS-SSIM.

Returns SSIM computed (higher is better).

Return type float

xiuminglib.metric.**compute_ci**(*data*, *level=0.95*)
Computes confidence interval.

Parameters

- **data** (*list(float)*) – Samples.
- **level** (*float, optional*) – Confidence level. Defaults to 0.95.

Returns One-sided interval (i.e., mean $\pm$ this number).

Return type float

3.1.12 xiuminglib.os module

xiuminglib.os.**call**(*cmd*, *cwd=None*, *wait=True*, *quiet=False*)
Executes a command in shell.

Parameters

- **cmd** (*str*) – Command to be executed.
- **cwd** (*str, optional*) – Directory to execute the command in. `None` means current directory.
- **wait** (*bool, optional*) – Whether to block until the call finishes.
- **quiet** (*bool, optional*) – Whether to print out the output stream (if any) and error stream (if error occured).

Returns

- **retcode** (*int*) – Command exit code. 0 means a successful call. Always `None` if not waiting for the command to finish.
- **stdout** (*str*) – Standard output stream. Always `None` if not waiting.
- **stderr** (*str*) – Standard error stream. Always `None` if not waiting.

Return type tuple

xiuminglib.os.**cp**(*src*, *dst*, *cns_parallel_copy=10*)
Copies files, possibly from/to the Google Colossus Filesystem.

Parameters

- **src** (*str*) – Source file or directory.
- **dst** (*str*) – Destination file or directory.
- **cns_parallel_copy** (*int*) – The number of files to be copied in parallel. Only effective when copying a directory from/to Colossus.

xiuminglib.os.**exists_isdir**(*path*)
Determines whether a path exists, and if so, whether it is a file or directory.

Supports Google Colossus (CNS) paths by using `gfile` (preferred for speed) or the `fileutil` CLI.

Parameters **path** (*str*) – A path.

Returns

- **exists** (*bool*) – Whether the path exists.

- **isdir** (*bool*) – Whether the path is a file or directory. `None` if the path doesn't exist.

Return type tuple

`xiuminglib.os.`**`fix_terminal`**`()`
Fixes messed up terminal.

`xiuminglib.os.`**`make_exp_dir`**`(`*directory*, *param_dict*, *rm_if_exists=False*`)`
Makes an experiment output folder by hashing the experiment parameters.

Parameters

- **directory** (*str*) – The made folder will be under this.
- **param_dict** (*dict*) – Dictionary of the parameters identifying the experiment. It is sorted by its keys, so different orders lead to the same hash.
- **rm_if_exists** (*bool, optional*) – Whether to remove the experiment folder if it already exists.

Writes

- The experiment parameters in `<directory>/<hash>/param.json`.

Returns The experiment output folder just made.

Return type str

`xiuminglib.os.`**`makedirs`**`(`*directory*, *rm_if_exists=False*`)`
Wraps `os.makedirs()` to support removing the directory if it alread exists.

Google Colossus-compatible: it tries to use `gfile` first for speed. This will fail if Blaze is not used, in which case it then falls back to using `fileutil` CLI as external process calls.

Parameters

- **directory** (*str*) –
- **rm_if_exists** (*bool, optional*) – Whether to remove the directory (and its contents) if it already exists.

`xiuminglib.os.`**`open_file`**`(`*path*, *mode*`)`
Opens a file.

Supports Google Colossus if `gfile` can be imported.

Parameters

- **path** (*str*) – Path to open.
- **mode** (*str*) – `'r'`, `'rb'`, `'w'`, or `'wb'`.

Returns File handle that can be used as a context.

`xiuminglib.os.`**`rm`**`(`*path*`)`
Removes a file or recursively a directory, with Google Colossus compatibility.

Parameters **path** (*str*) –

`xiuminglib.os.`**`sortglob`**`(`*directory*, *filename='*'*, *ext=None*, *ext_ignore_case=False*`)`
Globs and then sorts filenames, possibly ending with multiple extensions, in a directory.

Supports Google Colossus, by using `gfile` (preferred for speed) or the `fileutil` CLI when Blaze is not used (hence, `gfile` unavailable).

Parameters

- **directory** (*str*) – Directory to glob, e.g., `'/path/to/'`.
- **filename** (*str or tuple(str), optional*) – Filename pattern excluding extensions, e.g., `'img*'`.
- **ext** (*str or tuple(str), optional*) – Extensions of interest, e.g., `('png', 'jpg')`. `None` means no extension, useful for folders or files with no extension.
- **ext_ignore_case** (*bool, optional*) – Whether to ignore case for extensions.

Returns Sorted list of files globbed.

Return type list(str)

3.1.13 xiuminglib.sig module

xiuminglib.sig.**dct_1d_bases**(*n*)

Generates 1D discrete cosine transform (DCT) bases.

Bases are rows of Y, which is orthogonal: $Y^TY = YY^T = I$. The forward process (analysis) is $X = Yx$, and the inverse (synthesis) is $x = Y^{-1}X = Y^TX$. See *main()* for example usages and how this produces the same results as `scipy.fftpack.dct()` (with `type=2` and `norm='ortho'`).

Parameters **n** (*int*) – Signal length.

Returns Matrix whose i-th row, when dotted with signal (column) vector, gives the coefficient for the i-th DCT component. Of shape `(n, n)`.

Return type numpy.ndarray

xiuminglib.sig.**dct_2d_bases**(*h*, *w*)

Generates bases for 2D discrete cosine transform (DCT).

Bases are given in two matrices Y_h and Y_w. See *dct_1d_bases()* for their properties. Note that Y_w has already been transposed (hence, Y_hxY_w instead of $Y_hxY_w^T$ below).

Input image x should be transformed by both matrices (i.e., along both dimensions). Specifically, the analysis process is $X = Y_hxY_w$, and the synthesis process is $x = Y_h^TXY_w^T$. See *main()* for example usages.

Parameters

- **h** (*int*) – Image height.
- **w** –

Returns

- **dct_mat_h** (*numpy.ndarray*) – DCT matrix Y_h transforming rows of the 2D signal. Of shape `(h, h)`.
- **dct_mat_w** (*numpy.ndarray*) – Y_w transformingcolumns. Of shape `(w, w)`.

Return type tuple

xiuminglib.sig.**dct_2d_bases_vec**(*h*, *w*)

Generates bases stored in a single matrix, along whose height 2D frequencies get raveled.

Using the "vectorization + Kronecker product" trick: $\text{vec}(Y_hxY_w) = \left(Y_w^T \otimes Y_h\right)\text{vec}(x)$. So unlike *dct_2d_bases()*, this function generates a single matrix $Y = Y_w^T \otimes Y_h$, whose k-th row is the flattened (i, j)-th basis, where $k = wi + j$.

Input image x can be transformed with a single matrix multiplication. Specifically, the analysis process is $X = Y\,\text{vec}(x)$, and the synthesis process is $x = \text{unvec}(Y^TX)$. See *main()* for examples.

Warning: If you want to reconstruct the signal with only some (i.e., not all) bases, do not slice those rows out from Y and use only their coefficients. Instead, you should use the full Y matrix and set to zero the coefficients for the unused frequency components. See `main()` for examples.

Parameters

- **h** (*int*) – Image height.
- **w** –

Returns Matrix with flattened bases as rows. The k-th row, when `numpy.reshape()`'ed into `(h, w)`, is the :math:‘ (i, j)‘-th frequency component, where $k = wi + j$. Of shape `(h * w, h * w)`.

Return type numpy.ndarray

`xiuminglib.sig.`**`dft_1d_bases`**(*n*)

Generates 1D discrete Fourier transform (DFT) bases.

Bases are rows of Y, which is unitary ($Y^H Y = YY^H = I$, where Y^H is the conjugate transpose) and symmetric. The forward process (analysis) is $X = Yx$, and the inverse (synthesis) is $x = Y^{-1}X = Y^H X$. See `main()` for example usages.

Parameters **n** (*int*) – Signal length.

Returns Matrix whose i-th row, when dotted with signal (column) vector, gives the coefficient for the i-th Fourier component. Of shape `(n, n)`.

Return type numpy.ndarray

`xiuminglib.sig.`**`dft_2d_bases`**(*h*, *w*)

Generates bases for 2D discrete Fourier transform (DFT).

Bases are given in two matrices Y_h and Y_w. See `dft_1d_bases()` for their properties. Note that Y_w has already been transposed.

Input image x should be transformed by both matrices (i.e., along both dimensions). Specifically, the analysis process is $X = Y_h x Y_w$, and the synthesis process is $x = Y_h^H X Y_w^H$. See `main()` for example usages and how this produces the same results as `numpy.fft.fft2()` (with `norm='ortho'`).

See also:

From `numpy.fft` – `A[1:n/2]` contains the positive-frequency terms, and `A[n/2+1:]` contains the negative-frequency terms, in order of decreasingly negative frequency. For an even number of input points, `A[n/2]` represents both positive and negative Nyquist frequency, and is also purely real for real input. For an odd number of input points, `A[(n-1)/2]` contains the largest positive frequency, while `A[(n+1)/2]` contains the largest negative frequency.

Parameters

- **h** (*int*) – Image height.
- **w** –

Returns

- **dft_mat_h** (*numpy.ndarray*) – DFT matrix Y_h transforming rows of the 2D signal. Of shape `(h, h)`.
- **dft_mat_w** (*numpy.ndarray*) – Y_w transforming columns. Of shape `(w, w)`.

Return type tuple

xiuminglib.sig.**dft_2d_bases_vec**(*h*, *w*)

Generates bases stored in a single matrix, along whose height 2D frequencies get raveled.

Using the "vectorization + Kronecker product" trick: $\text{vec}(Y_h x Y_w) = \left(Y_w^T \otimes Y_h\right) \text{vec}(x)$. So unlike *dft_2d_bases()*, this function generates a single matrix $Y = Y_w^T \otimes Y_h$, whose k-th row is the flattened (i, j)-th basis, where $k = wi + j$.

Input image x can be transformed with a single matrix multiplication. Specifically, the analysis process is $X = Y\,\text{vec}(x)$, and the synthesis process is $x = \text{unvec}(Y^H X)$. See *main()* for examples.

Parameters

- **h** (*int*) – Image height.
- **w** –

Returns Complex matrix with flattened bases as rows. The k-th row, when `numpy.reshape()`'ed into `(h, w)`, is the (i, j)-th frequency component, where $k = wi + j$. Of shape `(h * w, h * w)`.

Return type numpy.ndarray

xiuminglib.sig.**dft_2d_freq**(*h*, *w*)

Gets 2D discrete Fourier transform (DFT) sample frequencies.

Parameters

- **h** (*int*) – Image height.
- **w** –

Returns

- **freq_h** (*numpy.ndarray*) – Sample frequencies, in cycles per pixel, along the height dimension. E.g., if `freq_h[i, j] == 0.5`, then the `(i, j)`-th component repeats every 2 pixels along the height dimension.
- **freq_w**

Return type tuple

xiuminglib.sig.**get_extrema**(*arr*, *top=True*, *n=1*, *n_std=None*)

Gets top (or bottom) N value(s) from an M-D array, with the option to ignore outliers.

Parameters

- **arr** (*array_like*) – Array, which will be flattened if high-D.
- **top** (*bool, optional*) – Whether to find the top or bottom N.
- **n** (*int, optional*) – Number of values to return.
- **n_std** (*float, optional*) – Definition of outliers to exclude, assuming Gaussian. `None` means assuming no outlier.

Returns

- **ind** (*tuple*) – Indices that give the extrema, M-tuple of arrays of N integers.
- **val** (*numpy.ndarray*) – Extremum values, i.e., `arr[ind]`.

Return type tuple

xiuminglib.sig.**main**(*test_id*)

Unit tests that can also serve as example usage.

xiuminglib.sig.**pca**(*data_mat*, *n_pcs=None*, *eig_method='scipy.sparse.linalg.eigsh'*)

Performs principal component (PC) analysis on data.

Via eigendecomposition of covariance matrix. See `main()` for example usages, including reconstructing data with top K PCs.

Parameters

- **data_mat** (*array_like*) – Data matrix of N data points in the M-D space, of shape M-by-N, where each column is a point.
- **n_pcs** (*int, optional*) – Number of top PCs requested. `None` means $M - 1$.
- **eig_method** (*str, optional*) – Method for eigendecomposition of the symmetric covariance matrix: `'numpy.linalg.eigh'` or `'scipy.sparse.linalg.eigsh'`.

Returns

- **pcvars** (*numpy.ndarray*) – PC variances (eigenvalues of covariance matrix) in descending order.
- **pcs** (*numpy.ndarray*) – Corresponding PCs (normalized eigenvectors), of shape M-by-`n_pcs`. Each column is a PC.
- **projs** (*numpy.ndarray*) – Data points centered and then projected to the `n_pcs`-D PC space. Of shape `n_pcs`-by-N. Each column is a point.
- **data_mean** (*numpy.ndarray*) – Mean that can be used to recover raw data. Of length M.

Return type tuple

xiuminglib.sig.**sh_bases_real**(*l*, *n_lat*, *coord_convention='colatitude-azimuth'*, *_check_orthonormality=False*)

Generates real spherical harmonics (SHs).

See `main()` for example usages, including how to do both analysis and synthesis the SHs.

Not accurate when `n_lat` is too small. E.g., orthonormality no longer holds when discretization is too coarse (small `n_lat`), as numerical integration fails to approximate the continuous integration.

Parameters

- **l** (*int*) – Up to which band (starting form 0). The number of harmonics is $(l + 1)^2$. In other words, all harmonics within each band ($-l \leq m \leq l$) are used.
- **n_lat** (*int*) – Number of discretization levels of colatitude (for colatitude-azimuth convention; $[0, \pi]$) or latitude (for latitude-longitude convention; $[-\frac{\pi}{2}, \frac{\pi}{2}]$). With the same step size, `n_azimuth` will be twice as big, since azimuth (in colatitude-azimuth convention; $[0, 2\pi]$) or latitude (in latitude-longitude convention; $[-\pi, \pi]$) spans 2π.
- **coord_convention** (*str, optional*) – Coordinate system convention to use: `'colatitude-azimuth'` or `'latitude-longitude'`. Colatitude-azimuth vs. latitude-longitude convention:

```
3D
                                    ^ z (colat = 0; lat = pi/2)
                                    |
          (azi = 3pi/2;             |
           lng = -pi/2)   ---------+---------> y (azi = pi/2;
                                ,'  |              lng = pi/2)
                              ,'    |
    (colat = pi/2, azi = 0;  x      | (colat = pi; lat = -pi/2)
```

(continues on next page)

(continued from previous page)

```
     lat = 0, lng = 0)

2D
    (0, 0)                                  (pi/2, 0)
       +------------>  (0, 2pi)                ^ lat
       |             azi                       |
       |                                       |
       |                        (0, -pi) -------+-------> (0, pi)
       v colat                                 |         lng
    (pi, 0)                                    |
                                           (-pi/2, 0)
```

- **_check_orthonormality** (*bool, optional, internal*) – Whether to check orthonormal or not.

Returns

- **ymat** (*numpy.ndarray*) – Matrix whose rows are spherical harmonics as generated by `scipy.special.sph_harm()`. When dotted with flattened image (column) vector weighted by `areas_on_unit_sphere`, the i-th row gives the coefficient for the i-th harmonics, where $i = l(l+1)+m$. The input signal (in the form of 2D image indexed by two angles) should be flattened with `numpy.ndarray.ravel()`, in row-major order: the row index varies the slowest, and the column index the quickest. Of shape `((l + 1) ** 2, 2 * n_lat ** 2)`.
- **areas_on_unit_sphere** (*numpy.ndarray*) – Area of the unit sphere covered by each sample point. This is proportional to sine of colatitude and has nothing to do with azimuth/longitude. Used as weights for discrete summation to approximate continuous integration. Necessary in SH analysis. Flattened also in row-major order. Of length `n_lat * (2 * n_lat)`.

Return type tuple

xiuminglib.sig.**smooth_1d**(*arr*, *win_size*, *kernel_type='half'*)

Smooths 1D signal.

Parameters

- **arr** (*array_like*) – 1D signal to smooth.
- **win_size** (*int*) – Size of the smoothing window. Use odd number.
- **kernel_type** (*str, optional*) – Kernel type: `'half'` (e.g., normalized $[2^{-2}, 2^{-1}, 2^{0}, 2^{-1}, 2^{-2}]$) or `'equal'` (e.g., normalized $[1, 1, 1, 1, 1]$).

Returns Smoothed 1D signal.

Return type numpy.ndarray

3.1.14 xiuminglib.tracker module

class xiuminglib.tracker.**LucasKanadeTracker**(*frames*, *pts*, *backtrack_thres=1*, *lk_params=None*)

Bases: `object`

Lucas Kanade Tracker.

frames

Grayscale.

Type list(numpy.array)

pts

Type numpy.array

lk_params

Type dict

backtrack_thres

Type float

tracks
Positions of tracks from the i-th to $(i+1)$-th frame. Arrays are of shape N-by-2.

```
+------------>
|       tracks[:, 1]
|
|
v tracks[:, 0]
```

Type list(numpy.array)

can_backtrack
Whether each track can be back-tracked to the previous frame. Arrays should be Boolean.

Type list(numpy.array)

is_lost
Whether each track is lost in this frame. Arrays should be Boolean.

Type list(numpy.array)

__init__(*frames*, *pts*, *backtrack_thres=1*, *lk_params=None*)

Parameters

- **frames** (*list(numpy.array)*) – Frame images in order. Arrays are either H-by-W or H-by-W-by-3, and will be converted to grayscale.
- **pts** (*array_like*) – Points to track in the first frame. Of shape N-by-2.

```
+------------>
|       pts[:, 1]
|
|
v pts[:, 0]
```

- **backtrack_thres** (*float, optional*) – Largest pixel deviation in the x or y direction of a successful backtrack.
- **lk_params** (*dict, optional*) – Keyword parameters for `cv2.calcOpticalFlowPyrLK()`.

run(*constrain=None*)
Runs tracking.

Parameters **constrain** (*function, optional*) – Function applied to tracks before being fed to the next round. It should take in an N-by-2 arrays as well as the current workspace (as a dictionary) and return another array.

vis(*out_dir*, *marker_bgr=(0, 0, 255)*)
Visualizes results.

Parameters

- **out_dir** (*str*) – Output directory.
- **marker_bgr** (*tuple, optional*) – Marker BGR color.

Writes

- Each frame with tracked points marked out.

Python Module Index

X

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

U

V

W

X