PyR3

Release 0.3.0

Krzysztof Wiśniewski

Oct 02, 2022

CONTENTS


$\mathbb{R}^3$

CHAPTER

ONE

OVERVIEW

The PyR3 package serves two purposes:

- provides blender in form of python package (bpy)
- contains useful shortcuts and abstractions over bpy API

This package is released under MIT license. Be aware that dependencies might be using different licenses.

1.1 Installation

PyR3 is available on Python Package Index and can be installed automatically with **pip**:

```
pip install PyR3
```

You can also install the in-development version from github with:

```
pip install https://github.com/Argmaster/pyr3/archive/main.zip
```

1.2 Complicated bpy requirement

Unlike previous releases, since 0.2.2 bpy is no longer automatically installed when importing PyR3, as this solution was not what's expected by typical developer.

Now to install bpy automatically you have to invoke **PyR3.install_bpy** module:

```
python -m PyR3.install_bpy
```

Or you can use install_bpy_lib() function from this module. After installing bpy it has to be manually uninstalled. It may happen that in future releases some uninstall script will be created, but for now manual removal is the only way.

1.3 Documentation

Documentation is available on-line at https://pyr3.readthedocs.io/

You can also build documentation yourself using tox:

```
git clone hhttps://github.com/Argmaster/pyr3.git
cd PyR3
tox -e docs
```

CHAPTER

TWO

INSTALLATION

PyR3 is officially available at Python Package Index, therefore you can use PIP to install it Type following into your command line:

```
pip install PyR3
```

and hit enter. PyR3 requires bpy (blender as python module) to run, but has no automatic way to install it on users PC. You can either install it manually or use PyR3.install_bpy to download and install it automatically. For later, see next section.

2.1 Installing bpy

PyR3.install_bpy script contained in this package can be used to install bpy for currently used python.

Appropriate bpy binaries will be automatically downloaded from here in form of tar.gz archive. Downloaded file will be placed in *side-packages/..* (folder containing side-packages).

However, if you provide appropriate archive (name is important) in this location manually, it will be used instead of downloading. Therefore no internet connection will be needed first time PyR3 gets invoked. After installation tar.gz file is deleted, regardless of its origin.

When initializer script is invoked, it unpacks files from tar.gz archive in appropriate places, which differ depending on operating system:

- on **Windows** 2.93 folder will end up next to python executable you are using (even if you are using virtual environment, its necessary to place it there) and rest of the files will be copied to *site-packages*,
- on **Linux** tar archive will be unpacked into *site-packages* folder.

2.2 Package Removal

To remove PyR3 from your python use:

```
pip uninstall PyR3
```

However if you have installed bpy with PyR3 (or any other way) above command wont remove it. Files which belong to bpy have to be deleted manually, and can be found in places described in Installing bpy section.

CHAPTER

THREE

USAGE

PyR3 is a utility library consisting of multiple semi-separate blocks

3.1 PyR3.shortcut

All files used as examples are contained in this repository, under corresponding relative paths.

This subpackage contains useful abstractions and shortcuts over blender API, which are used multiple times in other parts of PyR3, but it is mend also for public access and use.

3.1.1 Manipulating selection and active object

Listing 1: examples/shortcut/context/selection.py

```
# -*- coding: utf-8 -*-
from PyR3.shortcut.context import Objects
from PyR3.shortcut.mesh import addCube

# Deselects contents of current scene: default cube, light source and camera
Objects.deselect_all()
# Create new cube, uses shortcut from mesh module
cube = addCube()
# Select single object - our newly created cube
Objects.select(cube)
# prints Objects[bpy.data.objects['Cube.001']]
print(Objects.selected)

# selects everything in scene
Objects.select_all()
# Now Objects.selected is a longer sequence:
# Objects[bpy.data.objects['Cube'], bpy.data.objects['Light'],...]
print(Objects.selected)

# Lets now see whats currently active:
print(Objects.active)  # <bpy_struct, Object("Cube.001") at 0x4b39578>
# We can change it to light source existing in scene:
light_source = Objects.selected[1]
Objects.active = light_source
print(Objects.active)  # <bpy_struct, Object("Light") at 0x43a19b8>
```

(continues on next page)

(continued from previous page)

```
# We can also use contents of Object.selected to alter selection:
selected = Objects.selected
selected.pop()
selected.pop(0)
selected.select_only_contained()
# Now it will print only Objects[bpy.data.objects['Light'], bpy.data.objects['Camera']]
print(Objects.selected)
```

3.1.2 Using temporary selection

Listing 2: examples/shortcut/context/temporary_selection.py

```
# -*- coding: utf-8 -*-
from PyR3.shortcut.context import Objects, temporarily_selected

print(Objects.selected)  # Objects[bpy.data.objects['Cube']]
cube = Objects.selected[0]

with temporarily_selected(*Objects.all()):
    print(Objects.selected)
    # Objects[bpy.data.objects['Cube'], bpy.data.objects['Light'], bpy.data.objects['Camera
↪']]

print(Objects.selected)  # Objects[bpy.data.objects['Cube']]
```

3.1.3 Managing scenes

Listing 3: examples/shortcut/context/scene.py

```
# -*- coding: utf-8 -*-
from PyR3.shortcut.context import (
    delScene,
    getScene,
    listScenes,
    newScene,
    setScene,
    wipeScenes,
)

# First lets see our current scene:
print(getScene())  # <bpy_struct, Scene("Scene") at 0x41a0828>
# Now lets create new one:
newScene()
# And see list of all existing scenes
print(listScenes())  # [bpy.data.scenes['Scene'], bpy.data.scenes['Scene.001']]
# Then we can destroy _all_ of them and use new empty scene
wipeScenes()
print(listScenes())  # [bpy.data.scenes['Scene.001']]
```

(continues on next page)

(continued from previous page)

```
# You can also manually set current scene:
old_scene = getScene()
newScene()
new_scene = getScene()
print(getScene() == new_scene)  # True
setScene(old_scene)
print(getScene() == old_scene)  # True
# deletes current scene
delScene()
print(getScene() == new_scene)  # True
```

3.2 PyR3.factory

All files used as examples are contained in this repository, under corresponding relative paths.

This package provides MeshFactory class which should be subclassed to create specialized mesh factories and fields sub-package containing predefined field types that can be used in MeshFactory subclass.

3.2.1 Creating MeshFactory subclasses

Example below presents mesh factory which creates cube with size depending on integer field value, with custom name.

Listing 4: examples/factory/subclass.py

```
# -*- coding: utf-8 -*-
from PyR3.bpy import bpy
from PyR3.factory.fields.Number import Integer
from PyR3.factory.fields.String import String
from PyR3.factory.MeshFactory import MeshFactory
from PyR3.shortcut.mesh import addCube


class MeshFactorySubclass(MeshFactory):

    size = Integer(value_range=range(1, 5))
    name = String(default="name")

    def render(self):
        cube: bpy.types.Object = addCube(size=self.size)
        cube.name = self.name
        # cube was just created so it's already selected
```

3.3 PyR3.contrib

This subpackage contains first-party MeshFactory subclasses.

3.4 PyR3.meshlib

All files used as examples are contained in this repository, under corresponding relative paths.

LibraryManager class contained in this sub-package is designed to work as high-level tool to retrieving static models from specially prepared libraries.

3.4.1 How to create libraries

The core of every library is a `__lib__.yaml` file, containing all library metadata, including list of models within library. Library assets can be structured however you want them to, as long as you are able to provide paths to them, relative to `__lib__.yaml` file directory.

3.4.2 `__lib__.yaml` file contents

`__lib__.yaml` file is a static configuration file written in YAML. This file have to contain following keys in top-level dictionary:

- `version` currently only version 1.0.0 is a valid value
- `name` library name, It is highly recommended to make it as unique as possible
- `author` author's name / nick
- `description` library description
- `lib_version` library version, a semantic versioning compatible version string
- `model_list` list of dictionaries containing model metadata, explained below.

Model information

To define a model, you have to use dictionary, containing following set of keys:

- `hash` model SHA1 hash encoded as base64, if not of a valid length, will be recalculated and replaced
- `version` model version, a semantic versioning compatible version string
- `author` model author's name / nick
- `description` model description
- `file` file path, relative to folder containing `__lib__.yaml` file
- `tags` model tag list, later they can be used to find all models with matching tag, tags don't have to be unique
- `scale` indicates how the model is scaled in comparison to real size.
- **`icon` path to an icon file for this model, relative to folder containing `__lib__.yaml`**
  file

Example `__lib__.yaml` file:

Listing 5: tests/test_meshlib/test_lib/__lib__.yaml

```yaml
version: 1.0.0
name: Example Lib
author: Argmaster
description: ''
lib_version: 1.0.0
model_list:
- hash: e+kOrn6hL4tcJIHHwYWNLTbhzzY=
  version: 1.0.0
  author: Argmaster
  description: ''
  tags:
  - CommonTag
  - Example
  - Example2
  icon: __default_icon__
  file: subfolder/model2.glb
  scale: 10.0
- hash: +B4LrpYDjvu3t74iPTBsdYfBbx0=
  version: 1.0.0
  author: Argmaster
  description: ''
  tags:
  - CommonTag
  - Example
  - Example1
  icon: __default_icon__
  file: model1.glb
  scale: 1.0
```

3.4.3 Extending list of model tags as a user

It is possible to extend set of tags assigned to model described in `__lib__.yaml` file without modifying this file. You can achieve it by adding `__user__.yaml` file in the same directory as `__lib__.yaml`. `__user__.yaml` file contains YAML code, with top-level dictionary containing only `tags` key. This key maps to a nested dictionary with keys being hashes of models (as a base64 encoded strings). The values are also dictionaries with following keys:

- `tags` a list of string tags extending base tag set of a model
- `comment` a human readable comment

Example `__user__.yaml` file:

Listing 6: tests/test_meshlib/test_lib/__user__.yaml

```yaml
tags:
  +B4LrpYDjvu3t74iPTBsdYfBbx0=:
    comment: No special comment
    tags:
    - UserCustomTag
    - UserCustomTag2
    - CommonTag
```

(continues on next page)

(continued from previous page)

```
  e+kOrn6hL4tcJIHHwYWNLTbhzzY=:
    comment: No special comment
    tags:
    - UserCustomTag
    - UserCustomTag1
```

CHAPTER

FOUR

PYR3

4.1 PyR3 package

4.1.1 Subpackages

PyR3.contrib package

Subpackages

PyR3.contrib.factories package

Submodules

PyR3.contrib.factories.CapacitorCase module

class PyR3.contrib.factories.CapacitorCase.**CapacitorCase**(***MF_params*)

Bases: *MeshFactory*

Generates cylindrical meshes looking similar to electrolytic capacitor cases.

bevel_segments = Integer Field

bevel_width = Length Field

circle_vertices = Integer Field

h1 = Length Field

h2 = Length Field

h3 = Length Field

material = BSDF_Material Field

radius = Length Field

render(***kwargs*)

scale = Float Field

Module contents

Module contents

PyR3.factory package

Subpackages

PyR3.factory.fields package

Submodules

PyR3.factory.fields.Field module

class PyR3.factory.fields.Field.**Field**(*, *default: Optional[Any] = None*)

Bases: ABC

abstract clean_value(*value: Optional[Any] = None*) → Any

digest(*value: Optional[Any] = None*) → Any

get_default()

PyR3.factory.fields.Number module

class PyR3.factory.fields.Number.**Boolean**(*, *default: Optional[Any] = None*)

Bases: *Field*

clean_value(*value: Optional[Any] = None*) → Any

class PyR3.factory.fields.Number.**Float**(*, *default: Optional[float] = None*, *min: Optional[float] = None*, *max: Optional[float] = None*, *not_null: bool = False*)

Bases: *Field*

check_if_in_range(*parsed_float*)

clean_value(*value*)

class PyR3.factory.fields.Number.**Integer**(*, *default: Optional[int] = None*, *value_range: Optional[range] = None*, *not_null: bool = False*)

Bases: *Field*

check_if_in_range(*parsed_int*)

clean_value(*value*)

PyR3.factory.fields.Select module

class PyR3.factory.fields.Select.**Select**(**values*, *default: Optional[int] = None*)

Bases: *Field*

clean_value(*value: Optional[Any] = None*) → None

digest(*value: Optional[Any] = None*) → Any

PyR3.factory.fields.String module

class PyR3.factory.fields.String.**Regex**(*pattern: re.Pattern | str*, ***, *default: str = None*, *flags: int = 0*)

Bases: *String*

String field with possibility to use regular expression to check if string format is valid.

Parameters

- **pattern** (*re.Pattern or str*) – Regular expression patter. String will be automatically compiled.
- **default** (*str, optional*) – [description], defaults to None
- **flags** (*int, optional*) – regular expression flags, from re module, defaults to 0

Raises

TypeError – if pattern is neither str or re.Pattern.

class PyR3.factory.fields.String.**String**(***, *default: Optional[str] = None*, *min_length: Optional[int] = None*, *max_length: Optional[int] = None*)

Bases: *Field*

String factory field. You can specify length and default for it. Its value is always a string.

Parameters

- **default** (*str, optional*) – Default value, used if no value is provided. If None, exception will be raised if no value will be given, defaults to None
- **min_length** (*int, optional*) – Minimal length of string. Exception will be raised if requirement will not be fullfiled. Defaults to None
- **max_length** (*int, optional*) – Minimal length of string. Exception will be raised if requirement will not be fullfiled. Defaults to None

clean_value(*value*)

digest(*value: Optional[str] = None*) → str

Consumes value and returns cleaned string. Raises exception if requirements for string format are not met.

Parameters

value (*str, optional*) – value to consume, defaults to None

Returns

cleaned string.

Return type

str

PyR3.factory.fields.Struct module

class PyR3.factory.fields.Struct.**Struct**(**args*, ***kwargs*)

Bases: *Field*

Parent class allowing to create custom struct classes grouping other field types by subclassing Struct in body of MeshFactory or another Strut field.

Struct field value is a SimpleNamespace.

clean_value(*params: Optional[dict] = None*) → SimpleNamespace

Consumes dictionary of values and returns SimpleNamespace containing cleaned values of fields. Redundant params will be ignored. If a value is missing, None will be passed to coresponding field.

Parameters
params (*dict, optional*) – dictionary of values, defaults to None

Returns
namespace with cleaned values.

Return type
SimpleNamespace

get_default()

class PyR3.factory.fields.Struct.**StructNamespace**

Bases: *dict*

dict()

PyR3.factory.fields.Unit module

class PyR3.factory.fields.Unit.**Angle**(*, *output_unit: str = 'rad'*, *default: str | Number = None*)

Bases: *Length*

Accepts float with optional angle unit suffix. Unit suffix causes float value to be converted to value with unit denoted by *output_unit*.

Valid unit suffixes are:

- **rad** for radians
- **/ pi** for radians, multiplied by (3.14...)
- **deg** for degrees
- **“ / sec** for seconds of angle
- **‘ / min** for minutes of angle

Signs that doesn’t match anything are ignored and treated as separators.

parser = SuffixParser, suffixes: ['"', 'sec', "'", 'min', '°', 'deg', '', 'pi', 'rad', '']

class PyR3.factory.fields.Unit.**Length**(*, *output_unit: str = 'm'*, *default: str | Number = None*)

Bases: *Field*

Accepts float with optional length unit suffix. Unit suffix causes float value to be converted to value with unit denoted by *output_unit*.

Valid unit suffixes are:

- **mil** for mils
- **in** for inches
- **ft** for feets
- **mm** for millimeters
- **cm** for centimeters
- **dm** for decimeters
- **m** for meters

Signs that doesn't match anything are ignored and treated as separators.

clean_value(*value: str | Number*) → float

digest(*literal: str | Number = None*) → float

Returns total value contained in the literal in meters.

Parameters
literal (*Union[str, Number]*) – literal to consume or Number

Raises
- **TypeError** – If other type than str or Number is given.
- **KeyError** – If value is None and no default is given.

Returns
total in meters.

Return type
float

parser = SuffixParser, suffixes: ['mil', 'in', 'ft', 'mm', 'cm', 'dm', 'm', '']

Module contents

Submodules

PyR3.factory.MeshFactory module

class PyR3.factory.MeshFactory.**MeshFactory**(***MF_params*)

Bases: object

Base class for a mesh factory object.

Mesh factory requires __doc__, __author__ and __version__ to be defined in mesh factory subclass, otherwise class instantiation will fail. Mesh factory can (and should) make use of Fields (subclasses of Field class) to specify mesh factory customization params. See PyR3.factory.fields modules for first-party fields. To specify field just set class attribute to instance of Field subclass. See *MeshFactory usage*.

static build_external(*class_: str | MeshFactory, **params*) → *Objects*

static build_external_direct_map(*class_: str | MeshFactory, param_source: Mapping*) → *Objects*

prevent_autoselect()

render(***kwargs*)

PyR3.factory.MeshFactory.**getfields**(*mesh_factory:* MeshFactory) → Dict[str, *Field*]

Returns fields specified for given MeshFactory.

Parameters
mesh_factory (MeshFactory) – Object to fetch fields from.

Returns
dictionary of factory fields.

Return type
dict

PyR3.factory.MeshFactory.**import_factory**(*class_: str*)

Imports factory class from module.

Parameters
class (*str*) – python import name in form <python_module_import_path>.class

Raises
TypeError – Raised if requested class is not descendant of MeshFactory.

Module contents

PyR3.factory.**build_and_save**(*generator_type: str*, *class_: str* | MeshFactory, *params: dict*, *save_path: Path*)

PyR3.factory.**build_from_file**(*src_file: Path*, *save_path: Path*)

Build mesh using configuration from file. Later mesh is saved to save_path.

Parameters

- **src_file** (*Path*) – source configuration file.
- **save_path** (*Path*) – destination save path.

PyR3.factory.**build_python**(*class_: str* | MeshFactory, *params: dict*)

PyR3.shortcut package

Submodules

PyR3.shortcut.context module

class PyR3.shortcut.context.**Objects**(*iterable=()*, */*)

Bases: list

As a class, provides set of static functionalities for managing global selection and currently selected and active objects.

As a instance, is a container for objects with possibility to select or deselect object(s) contained inside.

active: Object = None

Currently active object. It can be set to change active object.

static all()

classmethod delete(**ob: Object*) → None

Delete object(s) ob. It keeps previously selected object(s) selected.

Parameters
ob (*Object*) – Object(s) to select

classmethod delete_all() → None

Deletes all selectable objects.

static deselect(**ob: Object*)

Deselect given object(s).

Parameters
ob (*Object*) – object(s) to deselect

static deselect_all()

Deselects all objects available in viewport.

deselect_contained()

Deselects elements contained in this sequence.

classmethod duplicate(**ob: Object*) → None

Duplicates object(s)

Parameters
ob (*Object*) – object(s) to duplicate

static inverse_selection()

only() → Object

Returns only element in sequence.

Raises
ValueError – If sequence is empty or has more than one element.

Returns
Only element from sequence

Return type
Object

static select(**ob: Object*)

Select given object(s).

Parameters
ob (*Object*) – object(s) to select

static select_all()

Selects all objects available in viewport.

select_contained()

Selects elements contained in this sequence.

classmethod select_only(**ob: Object*)

Deselects all objects and selects only given one(s).

Parameters
ob (*Object*) – Object(s) to select

select_only_contained()

Selects only elements contained in this sequence.

selected: *Objects*[Object]

List of currently selected objects. This property is read-only. Use methods to alter selection.

PyR3.shortcut.context.**cleanScene**() → None

Deletes current scene and creates new one.

Be aware that for total clean-up you should call wipeScenes() instead, as it destroys **ALL** scenes, not only current one, as cleanScene() does.

PyR3.shortcut.context.**delScene**() → None

Deletes currently used scene.

PyR3.shortcut.context.**getScene**() → Scene

Returns currently used scene.

Returns
Scene

Return type
bpy.types.Scene

PyR3.shortcut.context.**listScenes**() → List[Scene]

Returns list of existing scenes.

PyR3.shortcut.context.**newScene**() → None

Creates new Scene object and automatically sets it as currently used one.

PyR3.shortcut.context.**setScene**(*scene: Scene*) → None

Sets new scene to use.

Parameters
scene (*bpy.types.Scene*) – Scene

PyR3.shortcut.context.**temporarily_selected**(**ob: Object*)

For context manager usage, on enter selects only objects passed to constructor, on exit restores selection on previously selected objects.

PyR3.shortcut.context.**temporary_scene**()

Creates temporary scene and sets it as currently used. After exit, all objects selected in temporary scene are copied into previous scene and previous scene is set as currently used.

Yield
(new, old) scenes

Return type
Tuple[bpy.types.Scene, bpy.types.Scene]

PyR3.shortcut.context.**wipeScenes**() → None

Destroys all existing ones and creates new empty one.

PyR3.shortcut.edit module

class PyR3.shortcut.edit.**Edit**(*ob: Object*, **more: Object*, *active: Optional[Object] = None*)

Bases: object

class for automatic in-out switching Edit mode.

It is meant to be used as context manager with edited object being passed as param to constructor.

BMESH: BMesh = None

class **MeshCompList**(*iterable=()*, */*)

Bases: list

selected() → *MeshCompList*

bevel(*offset: float = 0.0*, *profile_type: Union[int, str] = 'SUPERELLIPSE'*, *offset_pct: float = 0.0*, *segments: int = 1*, *profile: float = 0.5*, *affect: Union[int, str] = 'EDGES'*, *clamp_overlap: bool = False*, *loop_slide: bool = True*, *mark_seam: bool = False*, *mark_sharp: bool = False*, *material: int = -1*, *harden_normals: bool = False*, *face_strength_mode: Union[int, str] = 'NONE'*, *miter_outer: Union[int, str] = 'SHARP'*, *miter_inner: Union[int, str] = 'SHARP'*, *spread: float = 0.1*, *vmesh_method: Union[int, str] = 'ADJ'*, *release_confirm: bool = False*)

Cut into selected items at an angle to create bevel or chamfer

Parameters

- **offset_type** (*Union[int, str]*) – Width Type, The method for determining the size of the bevel * OFFSET Offset, Amount is offset of new edges from original. * WIDTH Width, Amount is width of new face. * DEPTH Depth, Amount is perpendicular distance from original edge to bevel face. * PERCENT Percent, Amount is percent of adjacent edge length. * ABSOLUTE Absolute, Amount is absolute distance along adjacent edge.
- **offset** (*float*) – Width, Bevel amount
- **profile_type** (*Union[int, str]*) – Profile Type, The type of shape used to rebuild a beveled section * SUPERELLIPSE Superellipse, The profile can be a concave or convex curve. * CUSTOM Custom, The profile can be any arbitrary path between its endpoints.
- **offset_pct** (*float*) – Width Percent, Bevel amount for percentage method
- **segments** (*int*) – Segments, Segments for curved edge
- **profile** (*float*) – Profile, Controls profile shape (0.5 = round)
- **affect** (*Union[int, str]*) – Affect, Affect edges or vertices * VERTICES Vertices, Affect only vertices. * EDGES Edges, Affect only edges.
- **clamp_overlap** (*bool*) – Clamp Overlap, Do not allow beveled edges/vertices to overlap each other
- **loop_slide** (*bool*) – Loop Slide, Prefer sliding along edges to even widths
- **mark_seam** (*bool*) – Mark Seams, Mark Seams along beveled edges
- **mark_sharp** (*bool*) – Mark Sharp, Mark beveled edges as sharp
- **material** (*int*) – Material Index, Material for bevel faces (-1 means use adjacent faces)
- **harden_normals** (*bool*) – Harden Normals, Match normals of new faces to adjacent faces
- **face_strength_mode** (*Union[int, str]*) – Face Strength Mode, Whether to set face strength, and which faces to set face strength on * NONE None, Do not set face strength. *

NEW New, Set face strength on new faces only. * AFFECTED Affected, Set face strength on new and modified faces only. * ALL All, Set face strength on all faces.

- **miter_outer** (*Union[int, str]*) – Outer Miter, Pattern to use for outside of miters * SHARP Sharp, Outside of miter is sharp. * PATCH Patch, Outside of miter is squared-off patch. * ARC Arc, Outside of miter is arc.
- **miter_inner** (*Union[int, str]*) – Inner Miter, Pattern to use for inside of miters * SHARP Sharp, Inside of miter is sharp. * ARC Arc, Inside of miter is arc.
- **spread** (*float*) – Spread, Amount to spread arcs for arc inner miters
- **vmesh_method** (*Union[int, str]*) – Vertex Mesh Method, The method to use to create meshes at intersections * ADJ Grid Fill, Default patterned fill. * CUTOFF Cutoff, A cutoff at each profile's end before the intersection.
- **release_confirm** (*bool*) – Confirm on Release

collapse()

Collapse isolated edge and face regions, merging data such as UV's and vertex colors. This can collapse edge-rings as well as regions of connected faces into vertices

delete_edges()

Delete selected edges.

delete_faces()

Delete selected faces.

delete_vertices()

Delete selected vertices.

deselect_all()

Deselects whole mesh.

duplicate(*mode: int = 1*)

Duplicate selected.

edge_face_add()

Add an edge or face to selected

edges() → *MeshCompList*[BMEdge]

Provides access to edited object bmesh attribute holding reference to list of all edges of edited mesh.

Returns
List of edges.

Return type
MeshCompList[BMEdge]

extrude(*use_dissolve_ortho_edges: bool = False*, *mirror: bool = False*)

Extrude region of faces

Parameters

- **use_normal_flip** (*bool*) – Flip Normals
- **use_dissolve_ortho_edges** (*bool*) – Dissolve Orthogonal Edges
- **mirror** (*bool*) – Mirror Editing

extrude_individual_faces(*mirror: bool = False*)

Extrude individual edges only

Parameters

- **use_normal_flip** (*bool*) – Flip Normals
- **mirror** (*bool*) – Mirror Editing

extrude_repeat(*offset: List[float] = (0.0, 0.0, 0.0)*, *scale_offset: float = 1.0*)

Extrude selected vertices, edges or faces repeatedly

Parameters

- **steps** (*int*) – Steps
- **offset** (*List[float]*) – Offset, Offset vector
- **scale_offset** (*float*) – Scale Offset

faces() → *MeshCompList*[BMFace]

Provides access to edited object bmesh attribute holding reference to list of all faces of edited mesh.

Returns
List of faces.

Return type
MeshCompList[BMFace]

get_selected_vertices() → List[BMVert]

invert_selection()

Invertices selection of mesh components.

static isEditMode()

normals_make_consistent()

Make face and vertex normals point either outside or inside the mesh

Parameters
inside (*bool*) – Inside

ob: Object

remove_doubles(*use_unselected: bool = False*, *use_sharp_edge_from_normals: bool = False*)

Merge vertices based on their proximity

Parameters

- **threshold** (*float*) – Merge Distance, Maximum distance between elements to merge
- **use_unselected** (*bool*) – Unselected, Merge selected to other unselected vertices
- **use_sharp_edge_from_normals** (*bool*) – Sharp Edges, Calculate sharp edges using custom normal data (when available)

select_all()

Selects whole mesh.

select_edges(*condition: Callable[[Vector, Vector], bool]*)

Selects edges, when condition function returns true.

Parameters
condition (*Callable[[Vector, Vector], bool]*) – Test callable. It will be given edge vertice coordinate as parameter.

select_facing(*direction: Vector*) → *Edit*

select_vertices(*condition: Callable[[Vector], bool]*, *only: bool = False*)

Selects vertices, when condition function returns true.

Parameters
condition (*Callable[Vector], bool]*) – test callable. It will be given vertice coordinate as parameter.

smooth_faces()

vertices() → *MeshCompList*[BMVert]

Access to edited object bmesh vertice table.

Returns
Vertices

Return type
MeshCompList[BMVert]

exception PyR3.shortcut.edit.**OperationCancelled**

Bases: Exception

PyR3.shortcut.edit.**manual_set_edit_mode**()

PyR3.shortcut.edit.**manual_set_object_mode**()

PyR3.shortcut.io module

IO module provides import/export functions with ability to recognize file format from filename. It's limited but handy solution, hence available here. Recognized formats are:

- **GLB** : glTF Binary (.glb), Exports a single file, with all data packed in binary form.
- **GLTF** : glTF Embedded (.gltf), Exports a single file, with all data packed in JSON.
- **FBX** : Autodesk Filmbox (.fbx)
- **X3D** : Extensible 3D Graphics (.x3d)
- **OBJ** : Wavefront OBJ (.obj)
- **PLY** : Polygon File Format / Polygon File Format (.ply)
- **STL** : STL triangle mesh data (.stl)
- **BLEND / BLEND1** : Blender file format (.blend/.blend1) be aware that it causes to overwrite current scene on import.

PyR3.shortcut.io.**export_to**(*filepath: str*, ***kwargs*)

Export all objects into file. Format is determined from file extension. kwargs will be forwarded to bpy method call coresponding to selected format.

Parameters
filepath (*str*) – _Path to the file to export to.

Raises
KeyError – if format is not recognized.

PyR3.shortcut.io.**import_from**(*filepath: str*, ***kwargs*)

Import data from file. Format is determined from file extension. kwargs will be forwarded to bpy method call coresponding to selected format.

Parameters
filepath (*str*) – _Path to file to import.

Raises
KeyError – if format is not recognized.

PyR3.shortcut.material module

PyR3.shortcut.material.**apply_BSDF_material_params**(*ob: Optional[Object] = None*, *params: Optional[Dict[str, Any]] = None*) → str

Apply BSDF Node params to this type of node of given object ob. If ob is None, currently active object is used. If params is none, only result of calling this function is creation of new material for ob if it had no materials before.

Parameters

- **ob** (*Object, optional*) – Object, to which's material to apply params to, defaults to None
- **params** (*Dict[str, Any], optional*) – Dictionary of material params (listed in update_BSDF_node() function), defaults to None

Returns
Name of modified material.

Return type
str

PyR3.shortcut.material.**new_node_material**(*name: str = 'material'*)

PyR3.shortcut.material.**set_material**(*ob: Object*, *material: Material*)

PyR3.shortcut.material.**update_BSDF_node**(*material: Material*, *color: Optional[Tuple[float, float, float, float]] = None*, *subsurface: Optional[float] = None*, *subsurfaceRadius: Optional[Tuple[float, float, float]] = None*, *subsurfaceColor: Optional[Tuple[float, float, float, float]] = None*, *metallic: Optional[float] = None*, *specular: Optional[float] = None*, *specularTint: Optional[float] = None*, *roughness: Optional[float] = None*, *anisotropic: Optional[float] = None*, *anisotropicRotation: Optional[float] = None*, *sheen: Optional[float] = None*, *sheenTint: Optional[float] = None*, *clearcoat: Optional[float] = None*, *clearcoatRoughness: Optional[float] = None*, *IOR: Optional[float] = None*, *transmission: Optional[float] = None*, *transmissionRoughness: Optional[float] = None*, *emission: Optional[Tuple[float, float, float, float]] = None*, *emissionStrength: Optional[float] = None*, *alpha: Optional[float] = None*, ***, *autonormalize_color: bool = True*, *autonormalize_subsurfaceColor: bool = True*, *autonormalize_emission: bool = True*) → None

Updates default values in Principled BSDF node of material. None params are ignored and doesn't modify node.

Parameters

- **material** (*bpy.types.Material*) – Material to modify.

- **color** (*Color_T, optional*) – Diffuse or metal surface color.
- **subsurface** (*float, optional*) – Mix between diffuse and subsurface scattering. Rather than being a simple mix between Diffuse and Subsurface Scattering, it acts as a multiplier for the Subsurface Radius.
- **subsurfaceRadius** (*Tuple[float, float, float], optional*) – Average distance that light scatters below the surface. Higher radius gives a softer appearance, as light bleeds into shadows and through the object. The scattering distance is specified separately for the RGB channels, to render materials such as skin where red light scatters deeper. The X, Y and Z values are mapped to the R, G and B values, respectively.
- **subsurfaceColor** (*Color_T, optional*) – Subsurface scattering base color.
- **metallic** (*float, optional*) – Blends between a non-metallic and metallic material model. A value of 1.0 gives a fully specular reflection tinted with the base color, without diffuse reflection or transmission. At 0.0 the material consists of a diffuse or transmissive base layer, with a specular reflection layer on top.
- **specular** (*float, optional*) – Amount of dielectric specular reflection. Specifies facing (along normal) reflectivity in the most common 0 - 8% range.
- **specularTint** (*float, optional*) – Tints the facing specular reflection using the base color, while glancing reflection remains white.
- **roughness** (*float, optional*) – Specifies microfacet roughness of the surface for diffuse and specular reflection.
- **anisotropic** (*float, optional*) – Amount of anisotropy for specular reflection. Higher values give elongated highlights along the tangent direction; negative values give highlights shaped perpendicular to the tangent direction.
- **anisotropicRotation** (*float, optional*) – Rotates the direction of anisotropy, with 1.0 going full circle.
- **sheen** (*float, optional*) – Amount of soft velvet like reflection near edges, for simulating materials such as cloth.
- **sheenTint** (*float, optional*) – Mix between white and using base color for sheen reflection.
- **clearcoat** (*float, optional*) – Extra white specular layer on top of others. This is useful for materials like car paint and the like.
- **clearcoatRoughness** (*float, optional*) – Roughness of clearcoat specular.
- **IOR** (*float, optional*) – Index of refraction for transmission.
- **transmission** (*float, optional*) – Mix between fully opaque surface at zero and fully glass like transmission at one.
- **transmissionRoughness** (*float, optional*) – With GGX distribution controls roughness used for transmitted light.
- **emission** (*Color_T, optional*) – Light emission from the surface, like the Emission shader.
- **emissionStrength** (*float, optional*) – Strength of the emitted light. A value of 1.0 will ensure that the object in the image has the exact same color as the Emission Color, i.e. make it 'shadeless'.
- **alpha** (*float, optional*) – Controls the transparency of the surface, with 1.0 fully opaque. Usually linked to the Alpha output of an Image Texture node.

PyR3.shortcut.mesh module

Mesh operation shortcuts, including creation, bounding box calculations and more.

PyR3.shortcut.mesh.**addCircle**(*vertices: int = 32*, *radius: float = 1*, *fill_type: str | int = 'NOTHING'*, *location: List[float] = (0, 0, 0)*, *rotation: List[float] = (0, 0, 0)*, *scale: List[float] = (0, 0, 0)*) → Object

Shortcut for creating circle mesh. It returns created object.

Parameters

- **vertices** (*int, optional*) – number of vertices in arc, defaults to 32
- **radius** (*float, optional*) – circle radius, defaults to 1
- **fill_type** (*str, optional*) – face/no face to fill, defaults to "NOTHING"
- **location** (*List[float], optional*) – world location of circle mesh, defaults to (0, 0, 0)
- **rotation** (*List[float], optional*) – rotation of a mesh, defaults to (0, 0, 0)
- **scale** (*List[float], optional*) – scale of a mesh, defaults to (0, 0, 0)

Returns
newly created Object.

Return type
Object

PyR3.shortcut.mesh.**addCone**(*vertices: int = 32*, *radius1: float = 1*, *radius2: float = 0*, *depth: float = 2*, *end_fill_type: str | int = 'NGON'*, *location: List[float] = (0, 0, 0)*, *rotation: List[float] = (0, 0, 0)*, *scale: List[float] = (0, 0, 0)*) → Object

Shortcut for creating cone. It returns created object.

Parameters

- **vertices** (*int, optional*) – number of vertices in arc, defaults to 32
- **radius1** (*float, optional*) – radius 1, defaults to 1
- **radius2** (*float, optional*) – radius 2, defaults to 0
- **depth** (*float, optional*) – cone length, defaults to 2
- **end_fill_type** (*str, optional*) – face / no face at the end of cone mesh, defaults to "NGON"
- **location** (*List[float], optional*) – world location of center of cone, defaults to (0, 0, 0)
- **rotation** (*List[float], optional*) – rotation of a mesh, defaults to (0, 0, 0)
- **scale** (*List[float], optional*) – scale of a mesh, defaults to (0, 0, 0)

Returns
newly created Object.

Return type
Object

PyR3.shortcut.mesh.**addCube**(*size: float = 1.0*, *calc_uvs: bool = True*, *enter_editmode: bool = False*, *location: List[float] = (0.0, 0.0, 0.0)*, *rotation: List[float] = (0.0, 0.0, 0.0)*, *scale: List[float] = (1.0, 1.0, 1.0)*)

Shortcut for creating cube. It returns created object.

https://docs.blender.org/api/current/bpy.ops.mesh.html#bpy.ops.mesh.primitive_cube_add

Parameters

- **size** (*float*) – Size
- **calc_uvs** (*bool*) – Generate UVs, Generate a default UV map
- **enter_editmode** (*bool*) – Enter Edit Mode, Enter edit mode when adding this object
- **location** (*List[float]*) – Location, Location for the newly added object
- **rotation** (*List[float]*) – Rotation, Rotation for the newly added object
- **scale** (*List[float]*) – Scale, Scale for the newly added object

PyR3.shortcut.mesh.**addCylinder**(**args*, ***kwargs*)

PyR3.shortcut.mesh.**addGrid**(**args*, ***kwargs*)

Shortcut for creating grid. It returns created object.

PyR3.shortcut.mesh.**addIcoSphere**(**args*, ***kwargs*)

Shortcut for creating ico sphere. It returns created object.

PyR3.shortcut.mesh.**addPlane**(**args*, ***kwargs*)

Shortcut for creating plane. It returns created object.

PyR3.shortcut.mesh.**addTorus**(**args*, ***kwargs*)

Shortcut for creating torus. It returns created object.

PyR3.shortcut.mesh.**addUVSphere**(**args*, ***kwargs*)

Shortcut for creating uv sphere. It returns created object.

PyR3.shortcut.mesh.**boundingBoxCenterPoint**(*ob: Object*) → Vector

Calculates center of bounding box.

Parameters
ob (*Object*) – Object to calculate for.

Returns
Center point.

Return type
Vector

PyR3.shortcut.mesh.**boundingBoxPoints**(*ob: Object*) → float

Calculates object's bounding box.

Parameters
bpy_obj (*Object*) – Object to get bbox of.

Returns
List of bounding box points.

Return type
float

PyR3.shortcut.mesh.**containingSphereRadius**(*ob: Object*, *center: Optional[Vector] = None*) → float

Calculate radius of a sphere that bounding box can fit in.

Parameters

- **ob** (*Object*) – Object to calculate for
- **center** (*Vector, optional*) – Changes center from bbox center, defaults to None

Returns
radius

Return type
float

PyR3.shortcut.mesh.**continuous_edge**(*vertices: List[Tuple[float, float, float]]*)

PyR3.shortcut.mesh.**convert**(*ob: Object*, *target: str = 'MESH'*)

Convert object from one type to another.

Parameters

- **ob** (*Object*) – Object to transform
- **target** (*str, optional*) – target object type, defaults to "MESH"

PyR3.shortcut.mesh.**fromPyData**(*vertexData: List[Tuple[float, float, float]] = []*, *edgeData: List[Tuple[float, float]] = []*, *faceData: List[Tuple[float, ...]] = []*, ***, *mesh_name='mesh'*, *object_name='object'*) → Object

Creates new mesh object from python data.

Parameters

- **vertexData** (*List[Tuple[float, float, float]], optional*) – list of vertices, defaults to []
- **edgeData** (*List[Tuple[float, float]], optional*) – list of tuples of edges vertices indexes, defaults to []
- **faceData** (*List[Tuple[float, ...]], optional*) – list of tuples of faces edge indexes, defaults to []
- **mesh_name** (*str, optional*) – name for mesh data object, defaults to "mesh"
- **object_name** (*str, optional*) – name for mesh object, defaults to "object"

Returns
created object.

Return type
Object

PyR3.shortcut.mesh.**join**(*target: Object*, **rest: Object*)

Joins rest objects into target object. This will result in merging meshes into one object's data.

Parameters

- **target** (*Object*) – object to join rest into
- **rest** (*Object*) – other objects to join

PyR3.shortcut.modifiers module

class PyR3.shortcut.modifiers.**Array**(*master_object: Object*, *constant_offset_displace: Tuple[float, float, float] = (0, 0, 0)*, *count: int = 1*, *use_relative_offset: bool = False*, *use_constant_offset: bool = True*)

Bases: _Modifier

Array modifier wrapper.

For documentation over modifier parameters visit https://docs.blender.org/api/current/bpy.types.ArrayModifier.html

constant_offset_displace: Tuple[float, float, float] = (0, 0, 0)

count: int = 1

master_object: Object

use_constant_offset: bool = True

use_relative_offset: bool = False

class PyR3.shortcut.modifiers.**Bevel**(*master_object: Object*, *affect: str = 'EDGES'*, *offset_type: str = 'OFFSET'*, *width: float = 0.1*, *segments: int = 1*, *limit_method: str = 'NONE'*, *angle_limit: float = 0.5235987755982988*, *use_clamp_overlap: bool = True*)

Bases: _Modifier

Solidify modifier wrapper.

For documentation over modifier parameters visit https://docs.blender.org/api/current/bpy.types.BevelModifier.html

affect: str = 'EDGES'

angle_limit: float = 0.5235987755982988

limit_method: str = 'NONE'

master_object: Object

offset_type: str = 'OFFSET'

segments: int = 1

use_clamp_overlap: bool = True

width: float = 0.1

class PyR3.shortcut.modifiers.**Boolean**(*master_object: Object*, *object: Object*, *operation: str = 'DIFFERENCE'*, *solver: str = 'EXACT'*, *use_self: bool = False*)

Bases: _Modifier

Boolean Modifier wrapper.

For documentation over modifier parameters visit https://docs.blender.org/api/current/bpy.types.BooleanModifier.html

master_object: Object

object: Object

operation: str = 'DIFFERENCE'

solver: str = 'EXACT'

use_self: bool = False

class PyR3.shortcut.modifiers.**Decimate**(*master_object: Object*, *angle_limit: float = 0.0872665*, *decimate_type: str = 'COLLAPSE'*, *delimit: FrozenSet[str] = frozenset({'NORMAL'})*, *invert_vertex_group: bool = False*, *iterations: int = 0*, *ratio: float = 1.0*, *symmetry_axis: str = 'X'*, *use_collapse_triangulate: bool = False*, *use_dissolve_boundaries: bool = False*, *vertex_group: str = ''*, *vertex_group_factor: float = 1.0*)

Bases: _Modifier

Decimate modifier wrapper.

For documentation over modifier parameters visit https://docs.blender.org/api/current/bpy.types.DecimateModifier.html

angle_limit: float = 0.0872665

decimate_type: str = 'COLLAPSE'

delimit: FrozenSet[str] = frozenset({'NORMAL'})

invert_vertex_group: bool = False

iterations: int = 0

master_object: Object

ratio: float = 1.0

symmetry_axis: str = 'X'

use_collapse_triangulate: bool = False

use_dissolve_boundaries: bool = False

vertex_group: str = ''

vertex_group_factor: float = 1.0

class PyR3.shortcut.modifiers.**Remesh**(*master_object: Object*, *adaptivity: float = 0.0*, *mode: str = 'BLOCKS'*, *octree_depth: int = 4*, *scale: float = 0.9*, *sharpness: float = 1.0*, *threshold: float = 1.0*, *use_remove_disconnected: bool = True*, *use_smooth_shade: bool = False*, *voxel_size: float = 0.1*)

Bases: _Modifier

Remesh modifier wrapper.

For documentation over modifier parameters visit https://docs.blender.org/api/current/bpy.types.RemeshModifier.html

adaptivity: float = 0.0

master_object: Object

mode: str = 'BLOCKS'

octree_depth: int = 4

scale: float = 0.9

sharpness: float = 1.0

threshold: float = 1.0

use_remove_disconnected: bool = True

use_smooth_shade: bool = False

voxel_size: float = 0.1

class PyR3.shortcut.modifiers.**Solidify**(*master_object: Object*, *thickness: float = 0.01*, *offset: float = -1*, *use_even_offset: bool = False*, *use_quality_normals: bool = True*)

Bases: _Modifier

Solidify modifier wrapper.

For documentation over modifier parameters visit https://docs.blender.org/api/current/bpy.types.SolidifyModifier.html

master_object: Object

offset: float = -1

thickness: float = 0.01

use_even_offset: bool = False

use_quality_normals: bool = True

PyR3.shortcut.transform module

class PyR3.shortcut.transform.**ApplyProxyObject**(*ob:* Objects)

Bases: object

apply_all()

apply_rotation()

apply_scale()

apply_transform()

class PyR3.shortcut.transform.**Transform**

Bases: object

This class is a container for set of object transforming functions.

They all operate on global (currently selected) object(s).

static apply(*do_move: bool = False*, *do_rotation: bool = False*, *do_scale: bool = False*)
Apply the object's transformation to its data.

Parameters

- **use_move** (*bool, optional*) – applies move if true, defaults to False
- **use_rotation** (*bool, optional*) – applies rotation if true, defaults to False
- **use_scale** (*bool, optional*) – applies scale if true, defaults to False

static move(**args*, ***kwargs*)

static resize(**args*, ***kwargs*)

static rotate(**args*, ***kwargs*)

static scale(**args*, ***kwargs*)

PyR3.shortcut.transform.**move**(*vector: Tuple[float, float, float]*, ***kwargs*) → *ApplyProxyObject*
Move selected objects.

Parameters

- **vector** (*Tuple[float, float, float], optional*) – absolute coordinates to move to
- ****kwargs** – All from bpy.ops.transform.translate.

PyR3.shortcut.transform.**rotate**(*angle: float*, *orient_axis: str*, ***kwargs*) → *ApplyProxyObject*
Rotate selected objects around *orient_axis*

Parameters

- **angle** (*float, optional*) – rotation angle
- **orient_axis** (*str, optional*) – axis to rotate around, either "X", "Y" or "Z".
- ****kwargs** – All from bpy.ops.transform.rotate.

PyR3.shortcut.transform.**scale**(*scales: Tuple[float, float, float]*, ***kwargs*)
Scale (resize) selected objects.

Parameters

- **scales** (*tuple, optional*) – Tuple of scales for each axis, (x, y, z)
- ****kwargs** – All from bpy.ops.transform.resize.

Module contents

4.1.2 Submodules

4.1.3 PyR3.install_bpy module

4.1.4 Module contents

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

PyR3 could always use more documentation, whether as part of the official PyR3 docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Argmaster/pyr3/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *pyr3* for local development:

1. Fork pyr3 (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/pyr3.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

Running tests in parallel is highly discouraged, as some of them make use of shared environment.

CHAPTER

SIX

DEVELOPMENT GUIDELINES

There are some rules in how PyR3 should be developed, related to git branch structure, testing and building. We will make an attempt to describe them here.

6.1 Git repo management

In general this project is making use of a successfull Git branching model With partial automation with python scripts and some changes.

6.1.1 `main` and `develop`

In repo there are two branches that lives next to each other.

The `develop` branch contains all in-development code with all unstable features and changes.

On the other hand `main` branch always contains most recent **stable** code.

6.1.2 Feature branches

Feature branches always follows following naming convention with "feature-" prefix and unique feature name postfix. eg. "feature-generators", "feature-linux-support"

Also, those branches are always created from `develop` branch and they merge back into `develop`.

To be sure you wont mess anything in feature branch creation you can use python script available in this repo in scripts/fork_feature.py:

```
python -m scripts.fork_feature <feature-name>
```

"feature-" prefix is automatically added so you don't have to include in script call args.

After you finish working on your feature open a pull request on GitHub and either wait for request being accepted or if you have necessary rights, merge it yourself. Don't use scripts/merge_feature.py anymore, pull requests are better way of managing branch merging in this case.

6.1.3 Release branches (major and minor)

After adding a bunch of features to `develop`, make sure you have described them in `CHANGELOG.rst` on `develop` branch. Make sure version tag above feature description matches version tag you are planning to use for this release.

Release branch have to be drafted from `develop` and you should use `scripts/fork_release.py` for it unless you are willing to fix all the messed up version tags. Im sure you don't, so never create release branch manually via git.

```
python -m scripts.fork_release --major
```

or

```
python -m scripts.fork_release --minor
```

Using –patch is highly discouraged, for small changes us hotfix approach.

After you make sure release branch is **release ready** go to GitHub and create pull request to merge `release-x.y.z` into `main`. It will run bunch of tests. **You should wait for tests to succeed.**

Then, If you have enough permissions, you can accept merge.

Merge will cause additional merges to run to sync remote branches, so after release you have to locally pull changes to both `main` and `develop`. It will also automatically create release with release notes from `CHANGELOG.rst` file.

PyPI release is not automatically created.

6.1.4 Hotfix branches (patch)

Hotfix branches are used to hotfix already existing releases. Therefore they are forked from release branch patch corresponds to. Make sure you use `scripts/fork_hotfix.py` to create hotfix branch, as all version tags have to be updated, and for sure you don't want to do it manually.

```
python -m scripts.fork_hotfix <version-tag-to-fork-from>
```

eg.

```
python -m scripts.fork_hotfix v0.3.0
```

To merge this branch do the same as with release branch (see **Release branches (major and minor)**).

CHAPTER

SEVEN

AUTHORS

- Krzysztof Wiśniewski - https://github.com/Argmaster

CHAPTER

EIGHT

CHANGELOG

8.1 0.0.0 (2021-09-27)

- First release on PyPI.

8.2 0.1.0 (2021-10-01)

- Added Modifiers: Boolean, Array, Solidify and Bevel
- Added fromPyData()
- Improved documentation
- Added example files
- Added dark theme to docs

8.3 0.1.1 (2021-10-01)

- Hotfix of missing dependencies in package

8.4 0.1.2 (2021-10-01)

- Hotfix of export/import API

8.5 0.2.0 (2021-10-03)

- Added materials shortcuts
- Updated documentation
- Bpy is no longer automatically installed
- Bpy can be now installed via PyR3.install_bpy script

8.6 0.2.3 (2021-10-03)

- Updated documentation

8.7 0.3.0 (2021-10-21)

- Introduced new development pipeline
- Extendent usage documentation
- .blend1 files no longer can be imported/exported with shortcuts.io functions
- Added LibraryManager class for managing 3D component libraries
- Added LibraryObject class responsible for managing libraries
- Added LibraryInfoV1_0_0 and ModelInfoV1_0_0 classes for `__lib__.yaml` version 1.0.0 files validation
- Added way to extend set tags of a model from `__lib__.yaml` - via `__user__.yaml`
- Added documentation for newest features
- Added MeshProject class and project configuration convention
- Added PlaceFile class which can parse place file and convert it into MeshProject file
- Added PyR3.construct CLI for operating on MeshProject files
- Added PyR3.meshlib CLI for operating on mesh libraries

8.8 0.4.0 (2021-11-08)

- Updated implementation of PyR3.contrib.factories.CapacitorCase
- Added API and CLI for rendering single models using MeshFactories
- Added Remesh modifier class
- Added SCurve MeshFactory subclass
- Added CapacitorCase and SCurve MeshFactory subclasses
- Added BSDF_Material, Color, HeterotypeSequence and HomotypeSequence Field types
- Added PyR3.shortcut.material.apply_BSDF_material_params() function for high level applying of material params
- Added ApplyProxyObject class for applying transforms to held objects
- All functions performing transformations now returns ApplyProxyObject object
- All methods of PyR3.shortcut.transform.Transform class methods are deprecated and will be removed in 1.0 update, use corresponding functions instead
- MeshFactory.render automatically selects everything in viewport, to prevent it call MeshFactory.prevent_autoselect()
- Fix stacking normalization of colors in nested BSDF_Materials when calling generators from generators
- Color field no longer does normalization by default
- PyR3.shortcut.material.update_BSDF_node now does auto normalization of color params

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W