Mesh(ID)

Mesh Data

The mesh data is accessed in object mode and intended for compact storage, for more flexible mesh editing from python see `bmesh`.

Blender stores 4 main arrays to define mesh geometry.

- `Mesh.vertices` (3 points in space)
- `Mesh.edges` (reference 2 vertices)
- `Mesh.loops` (reference a single vertex and edge)
- `Mesh.polygons`: (reference a range of loops)

Each polygon references a slice in the loop array, this way, polygons do not store vertices or corner data such as UVs directly, only a reference to loops that the polygon uses.

`Mesh.loops`, `Mesh.uv_layers` `Mesh.vertex_colors` are all aligned so the same polygon loop indices can be used to find the UVs and vertex colors as with as the vertices.

To compare mesh API options see: NGons and Tessellation Faces

This example script prints the vertices and UVs for each polygon, assumes the active object is a mesh with UVs.

```
import bpy

me = bpy.context.object.data
uv_layer = me.uv_layers.active.data

for poly in me.polygons:
    print("Polygon index: {:d}, length: {:d}".format(poly.index, poly.loop_total))

    # range is used here to show how the polygons reference loops,
    # for convenience 'poly.loop_indices' can be used instead.
    for loop_index in range(poly.loop_start, poly.loop_start + poly.loop_total):
        print("    Vertex: {:d}".format(me.loops[loop_index].vertex_index))
        print("    UV: {!r}".format(uv_layer[loop_index].uv))
```

base classes — `bpy_struct`, `ID`

class bpy.types.**Mesh(ID)**

Mesh data-block defining geometric surfaces

animation_data

Animation data for this data-block

TYPE:

`AnimData`, (readonly)

attributes

Geometry attributes

TYPE:

`AttributeGroupMesh` `bpy_prop_collection` of `Attribute`, (readonly)

auto_texspace

Adjust active object’s texture space automatically when transforming object

TYPE:

boolean, default True

color_attributes

Geometry color attributes

TYPE:

`AttributeGroupMesh` `bpy_prop_collection` of `Attribute`, (readonly)

corner_normals

The “slit” normal direction of each face corner, influenced by vertex normals, sharp faces, sharp edges, and custom normals. May be empty.

TYPE:

`bpy_prop_collection` of `MeshNormalValue`, (readonly)

cycles

Cycles mesh settings

TYPE:

`CyclesMeshSettings`, (readonly)

edges

Edges of the mesh

TYPE:

`MeshEdges` `bpy_prop_collection` of `MeshEdge`, (readonly)

has_custom_normals

True if there are custom split normals data in this mesh

TYPE:

boolean, default False, (readonly)

is_editmode

True when used in editmode

TYPE:

boolean, default False, (readonly)

loop_triangle_polygons

The face index for each loop triangle

TYPE:

`bpy_prop_collection` of `ReadOnlyInteger`, (readonly)

loop_triangles

Tessellation of mesh polygons into triangles

TYPE:

`MeshLoopTriangles` `bpy_prop_collection` of `MeshLoopTriangle`, (readonly)

loops

Loops of the mesh (face corners)

TYPE:

`MeshLoops` `bpy_prop_collection` of `MeshLoop`, (readonly)

materials

materials

TYPE:
`IDMaterials` `bpy_prop_collection` of `Material`, (readonly)

normals_domain

The attribute domain that gives enough information to represent the mesh's normals

TYPE:
enum in ['POINT', 'FACE', 'CORNER'], default 'FACE', (readonly)

polygon_normals

The normal direction of each face, defined by the winding order and position of its vertices

TYPE:
`bpy_prop_collection` of `MeshNormalValue`, (readonly)

polygons

Polygons of the mesh

TYPE:
`MeshPolygons` `bpy_prop_collection` of `MeshPolygon`, (readonly)

remesh_mode

- `VOXEL` Voxel – Use the voxel remesher.
- `QUAD` Quad – Use the quad remesher.

TYPE:
enum in ['VOXEL', 'QUAD'], default 'VOXEL'

remesh_voxel_adaptivity

Reduces the final face count by simplifying geometry where detail is not needed, generating triangles. A value greater than 0 disables Fix Poles

TYPE:
float in [0, 1], default 0.0

remesh_voxel_size

Size of the voxel in object space used for volume evaluation. Lower values preserve finer details.

TYPE:
float in [0.0001, inf], default 0.1

shape_keys

TYPE:
`Key`, (readonly)

skin_vertices

All skin vertices

TYPE:
`bpy_prop_collection` of `MeshSkinVertexLayer`, (readonly)

texco_mesh

Derive texture coordinates from another mesh

TYPE:
`Mesh`

texspace_location

Texture space location

TYPE:

mathutils.Vector of 3 items in [-inf, inf], default (0.0, 0.0, 0.0)

texspace_size

Texture space size

TYPE:

mathutils.Vector of 3 items in [-inf, inf], default (1.0, 1.0, 1.0)

texture_mesh

Use another mesh for texture indices (vertex indices must be aligned)

TYPE:

Mesh

total_edge_sel

Selected edge count in editmode

TYPE:

int in [0, inf], default 0, (readonly)

total_face_sel

Selected face count in editmode

TYPE:

int in [0, inf], default 0, (readonly)

total_vert_sel

Selected vertex count in editmode

TYPE:

int in [0, inf], default 0, (readonly)

use_auto_texspace

Adjust active object’s texture space automatically when transforming object

TYPE:

boolean, default True

use_mirror_topology

Use topology based mirroring (for when both sides of mesh have matching, unique topology)

TYPE:

boolean, default False

use_mirror_vertex_groups

Mirror the left/right vertex groups when painting. The symmetry axis is determined by the symmetry settings.

TYPE:

boolean, default True

use_mirror_x

Enable symmetry in the X axis

TYPE:

boolean, default False

boolean, default False

use_mirror_y

Enable symmetry in the Y axis

TYPE:

boolean, default False

use_mirror_z

Enable symmetry in the Z axis

TYPE:

boolean, default False

use_paint_bone_selection

Bone selection during painting

TYPE:

boolean, default True

use_paint_mask

Face selection masking for painting

TYPE:

boolean, default False

use_paint_mask_vertex

Vertex selection masking for painting

TYPE:

boolean, default False

use_remesh_fix_poles

Produces fewer poles and a better topology flow

TYPE:

boolean, default False

use_remesh_preserve_attributes

Transfer all attributes to the new mesh

TYPE:

boolean, default False

use_remesh_preserve_volume

Projects the mesh to preserve the volume and details of the original mesh

TYPE:

boolean, default False

uv_layer_clone

UV loop layer to be used as cloning source

TYPE:

`MeshUVLoopLayer`

uv_layer_clone_index

Clone UV loop layer index

TYPE:

int in [0, inf], default 0

uv_layer_stencil

UV loop layer to mask the painted area

TYPE:

`MeshUVLoopLayer`

uv_layer_stencil_index

Mask UV loop layer index

TYPE:

int in [0, inf], default 0

uv_layers

All UV loop layers

TYPE:

`UVLoopLayers` `bpy_prop_collection` of `MeshUVLoopLayer`, (readonly)

vertex_colors

Legacy vertex color layers. Deprecated, use color attributes instead.

TYPE:

`LoopColors` `bpy_prop_collection` of `MeshLoopColorLayer`, (readonly)

vertex_normals

The normal direction of each vertex, defined as the average of the surrounding face normals

TYPE:

`bpy_prop_collection` of `MeshNormalValue`, (readonly)

vertices

Vertices of the mesh

TYPE:

`MeshVertices` `bpy_prop_collection` of `MeshVertex`, (readonly)

edge_creases

Edge crease values for subdivision surface, corresponding to the “crease_edge” attribute.

(readonly)

edge_keys

(readonly)

vertex_creases

Vertex crease values for subdivision surface, corresponding to the “crease_vert” attribute.

(readonly)

vertex_paint_mask

Mask values for sculpting and painting, corresponding to the “.sculpt_mask” attribute.

(readonly)

transform(matrix, *, shape_keys=False)

Transform mesh vertices by a matrix (Warning: inverts normals if matrix is negative)

PARAMETERS:

- **matrix** (`mathutils.Matrix` of 4 * 4 items in [-inf, inf]) – Matrix
- **shape_keys** (*boolean, (optional)*) – Transform Shape Keys

flip_normals()

Invert winding of all polygons (clears tessellation, does not handle custom normals)

set_sharp_from_angle(*, angle=3.14159)

Reset and fill the "sharp_edge" attribute based on the angle of faces neighboring manifold edges

PARAMETERS:

angle (*float in [0, 3.14159], (optional)*) – Angle, Angle between faces beyond which edges are marked sharp

split_faces()

Split faces based on the edge angle

calc_tangents(*, uvmap='')

Compute tangents and bitangent signs, to be used together with the split normals to get a complete tangent space for normal mapping (split normals are also computed if not yet present)

PARAMETERS:

uvmap (*string, (optional, never None)*) – Name of the UV map to use for tangent space computation

free_tangents()

Free tangents

calc_loop_triangles()

Calculate loop triangle tessellation (supports editmode too)

calc_smooth_groups(*, use_bitflags=False)

Calculate smooth groups from sharp edges

PARAMETERS:

use_bitflags (*boolean, (optional)*) – Produce bitflags groups instead of simple numeric values

RETURNS:

poly_groups, Smooth Groups, int array of 1 items in [-inf, inf]

groups, Total number of groups, int in [0, inf]

RETURN TYPE:

(int array of 1 items in [-inf, inf], int in [0, inf])

normals_split_custom_set(normals)

Define custom split normals of this mesh (use zero-vectors to keep auto ones)

PARAMETERS:

normals (*float multi-dimensional array of 1 * 3 items in [-1, 1]*) – Normals

normals_split_custom_set_from_vertices(normals)

Define custom split normals of this mesh, from vertices' normals (use zero-vectors to keep auto ones)

PARAMETERS:

normals (*float multi-dimensional array of 1 * 3 items in [-1, 1]*) – Normals

update(*, calc_edges=False, calc_edges_loose=False)

update

PARAMETERS:

- **calc_edges** (*boolean, (optional)*) – Calculate Edges, Force recalculation of edges
- **calc_edges_loose** (*boolean, (optional)*) – Calculate Loose Edges, Calculate the loose state of each edge

update_gpu_tag()

update_gpu_tag

unit_test_compare(*, mesh=None, threshold=7.1526e-06)

unit_test_compare

PARAMETERS:

- **mesh** (`Mesh`, (optional)) – Mesh to compare to
- **threshold** (*float in [0, inf], (optional)*) – Threshold, Comparison tolerance threshold

RETURNS:

Return value, String description of result of comparison

RETURN TYPE:

string, (never None)

clear_geometry()

Remove all geometry from the mesh. Note that this does not free shape keys or materials.

validate(*, verbose=False, clean_customdata=True)

Validate geometry, return True when the mesh has had invalid geometry corrected/removed

PARAMETERS:

- **verbose** (*boolean, (optional)*) – Verbose, Output information about the errors found
- **clean_customdata** (*boolean, (optional)*) – Clean Custom Data, Remove temp/cached custom-data layers, like e.g. normals…

RETURNS:

Result

RETURN TYPE:

boolean

validate_material_indices()

Validate material indices of polygons, return True when the mesh has had invalid indices corrected (to default 0)

RETURNS:

Result

RETURN TYPE:

boolean

count_selected_items()

Return the number of selected items (vert, edge, face)

RETURNS:

Result

RETURN TYPE:

int array of 3 items in [0, inf]

edge_creases_ensure()

edge_creases_remove()

from_pydata(vertices, edges, faces, shade_flat=True)

Make a mesh from a list of vertices/edges/faces Until we have a nicer way to make geometry, use this.

PARAMETERS:

- **vertices** (*Iterable[Sequence[float]]*) – float triplets each representing (X, Y, Z) eg: [(0.0, 1.0, 0.5), …].
- **edges** (*Iterable[Sequence[int]]*) –

  int pairs, each pair contains two indices to the *vertices* argument. eg: [(1, 2), …]

  When an empty iterable is passed in, the edges are inferred from the polygons.
- **faces** (*Iterable[Sequence[int]]*) – iterator of faces, each faces contains three or more indices to the *vertices* argument. eg: [(5, 6, 8, 9 (1, 2, 3), …]

Warning

Invalid mesh data *(out of range indices, edges with matching indices, 2 sided faces… etc)* are **not** prevented. If the data used for mesh creation isn't known to be valid, run `Mesh.validate` after this function.

shade_flat()

Render and display faces uniform, using face normals, setting the "sharp_face" attribute true for every face

shade_smooth()

Render and display faces smooth, using interpolated vertex normals, removing the "sharp_face" attribute

vertex_creases_ensure()

vertex_creases_remove()

vertex_paint_mask_ensure()

vertex_paint_mask_remove()

classmethod bl_rna_get_subclass(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The RNA type or default when not found.

RETURN TYPE:

`bpy.types.Struct` subclass

classmethod bl_rna_get_subclass_py(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The class or default when not found.

RETURN TYPE:

type

Inherited Properties

- `bpy_struct.id_data`
- `ID.name`
- `ID.name_full`
- `ID.is_missing`
- `ID.is_runtime_data`

- ID.name_full
- ID.id_type
- ID.session_uid
- ID.is_evaluated
- ID.original
- ID.users
- ID.use_fake_user
- ID.use_extra_user
- ID.is_embedded_data
- ID.is_editable
- ID.tag
- ID.is_library_indirect
- ID.library
- ID.library_weak_reference
- ID.asset_data
- ID.override_library
- ID.preview

Inherited Functions

- bpy_struct.as_pointer
- bpy_struct.driver_add
- bpy_struct.driver_remove
- bpy_struct.get
- bpy_struct.id_properties_clear
- bpy_struct.id_properties_ensure
- bpy_struct.id_properties_ui
- bpy_struct.is_property_hidden
- bpy_struct.is_property_overridable_library
- bpy_struct.is_property_readonly
- bpy_struct.is_property_set
- bpy_struct.items
- bpy_struct.keyframe_delete
- bpy_struct.keyframe_insert
- bpy_struct.keys
- bpy_struct.path_from_id
- bpy_struct.path_resolve
- bpy_struct.pop
- bpy_struct.property_overridable_library_set
- bpy_struct.property_unset
- bpy_struct.type_recast
- bpy_struct.values
- ID.rename
- ID.evaluated_get
- ID.copy
- ID.asset_mark
- ID.asset_clear
- ID.asset_generate_preview
- ID.override_create
- ID.override_hierarchy_create
- ID.user_clear
- ID.user_remap
- ID.make_local
- ID.user_of_id
- ID.animation_data_create
- ID.animation_data_clear
- ID.update_tag
- ID.preview_ensure
- ID.bl_rna_get_subclass
- ID.bl_rna_get_subclass_py

References

- bpy.context.mesh
- BlendData.meshes
- BlendDataMeshes.new
- BlendDataMeshes.new_from_object
- BlendDataMeshes.remove
- Mesh.texco_mesh
- Mesh.texture_mesh
- Mesh.unit_test_compare
- Object.to_mesh