BMesh Operators (bmesh.ops)

This module gives access to low level bmesh operations.

Most operators take input and return output, they can be chained together to perform useful operations.

Operator Example

This script shows how operators can be used to model a link of a chain.

```
# This script uses bmesh operators to make 2 links of a chain.

import bpy
import bmesh
import math
import mathutils

# Make a new BMesh
bm = bmesh.new()

# Add a circle XXX, should return all geometry created, not just verts.
bmesh.ops.create_circle(
    bm,
    cap_ends=False,
    radius=0.2,
    segments=8)


# Spin and deal with geometry on side 'a'
edges_start_a = bm.edges[:]
geom_start_a = bm.verts[:] + edges_start_a
ret = bmesh.ops.spin(
    bm,
    geom=geom_start_a,
    angle=math.radians(180.0),
    steps=8,
    axis=(1.0, 0.0, 0.0),
    cent=(0.0, 1.0, 0.0))
edges_end_a = [ele for ele in ret["geom_last"]
               if isinstance(ele, bmesh.types.BMEdge)]
del ret


# Extrude and create geometry on side 'b'
ret = bmesh.ops.extrude_edge_only(
    bm,
    edges=edges_start_a)
geom_extrude_mid = ret["geom"]
del ret


# Collect the edges to spin XXX, 'extrude_edge_only' could return this.
verts_extrude_b = [ele for ele in geom_extrude_mid
                   if isinstance(ele, bmesh.types.BMVert)]
```

```
edges_extrude_b = [ele for ele in geom_extrude_mid
                   if isinstance(ele, bmesh.types.BMEdge) and ele.is_boundary]
bmesh.ops.translate(
    bm,
    verts=verts_extrude_b,
    vec=(0.0, 0.0, 1.0))


# Create the circle on side 'b'
ret = bmesh.ops.spin(
    bm,
    geom=verts_extrude_b + edges_extrude_b,
    angle=-math.radians(180.0),
    steps=8,
    axis=(1.0, 0.0, 0.0),
    cent=(0.0, 1.0, 1.0))
edges_end_b = [ele for ele in ret["geom_last"]
               if isinstance(ele, bmesh.types.BMEdge)]
del ret


# Bridge the resulting edge loops of both spins 'a & b'
bmesh.ops.bridge_loops(
    bm,
    edges=edges_end_a + edges_end_b)


# Now we have made a links of the chain, make a copy and rotate it
# (so this looks something like a chain)

ret = bmesh.ops.duplicate(
    bm,
    geom=bm.verts[:] + bm.edges[:] + bm.faces[:])
geom_dupe = ret["geom"]
verts_dupe = [ele for ele in geom_dupe if isinstance(ele, bmesh.types.BMVert)]
del ret

# position the new link
bmesh.ops.translate(
    bm,
    verts=verts_dupe,
    vec=(0.0, 0.0, 2.0))
bmesh.ops.rotate(
    bm,
    verts=verts_dupe,
    cent=(0.0, 1.0, 0.0),
    matrix=mathutils.Matrix.Rotation(math.radians(90.0), 3, 'Z'))

# Done with creating the mesh, simply link it into the scene so we can see it

# Finish up, write the bmesh into a new mesh
me = bpy.data.meshes.new("Mesh")
bm.to_mesh(me)
bm.free()
```

```
# Add the mesh to the scene
obj = bpy.data.objects.new("Object", me)
bpy.context.collection.objects.link(obj)

# Select and make active
bpy.context.view_layer.objects.active = obj
obj.select_set(True)
```

bmesh.ops.**smooth_vert(bm, verts=[], factor=0, mirror_clip_x=False, mirror_clip_y=False, mirror_clip_z=False, clip_dist=0, use_axis_x=False, use_axis_y=False, use_axis_z=False)**

Vertex Smooth.

Smooths vertices by using a basic vertex averaging scheme.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **factor** (*float*) – smoothing factor
- **mirror_clip_x** (*bool*) – set vertices close to the x axis before the operation to 0
- **mirror_clip_y** (*bool*) – set vertices close to the y axis before the operation to 0
- **mirror_clip_z** (*bool*) – set vertices close to the z axis before the operation to 0
- **clip_dist** (*float*) – clipping threshold for the above three slots
- **use_axis_x** (*bool*) – smooth vertices along X axis
- **use_axis_y** (*bool*) – smooth vertices along Y axis
- **use_axis_z** (*bool*) – smooth vertices along Z axis

bmesh.ops.**smooth_laplacian_vert(bm, verts=[], lambda_factor=0, lambda_border=0, use_x=False, use_y=False, use_z=False, preserve_volume=False)**

Vertex Smooth Laplacian.

Smooths vertices by using Laplacian smoothing propose by. Desbrun, et al. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **lambda_factor** (*float*) – lambda param
- **lambda_border** (*float*) – lambda param in border
- **use_x** (*bool*) – Smooth object along X axis
- **use_y** (*bool*) – Smooth object along Y axis
- **use_z** (*bool*) – Smooth object along Z axis
- **preserve_volume** (*bool*) – Apply volume preservation after smooth

bmesh.ops.**recalc_face_normals(bm, faces=[])**

Right-Hand Faces.

Computes an “outside” normal for the specified input faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces

bmesh.ops.**planar_faces(bm, faces=[], iterations=0, factor=0)**

Planar Faces.

Iteratively flatten faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input geometry.
- **iterations** (*int*) – Number of times to flatten faces (for when connected faces are used)
- **factor** (*float*) – Influence for making planar each iteration

RETURNS:

- `geom` : output slot, computed boundary geometry.

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**region_extend(bm, geom=[], use_contract=False, use_faces=False, use_face_step=False)**

Region Extend.

used to implement the select more/less tools. this puts some geometry surrounding regions of geometry in geom into geom.out.

if use_faces is 0 then geom.out spits out verts and edges, otherwise it spits out faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **use_contract** (*bool*) – find boundary inside the regions, not outside.
- **use_faces** (*bool*) – extend from faces instead of edges
- **use_face_step** (*bool*) – step over connected faces

RETURNS:

- `geom` : output slot, computed boundary geometry.

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**rotate_edges(bm, edges=[], use_ccw=False)**

Edge Rotate.

Rotates edges topologically. Also known as “spin edge” to some people. Simple example: *[/] becomes [\] then [/]*.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **use_ccw** (*bool*) – rotate edge counter-clockwise if true, otherwise clockwise

RETURNS:

- `edges` : newly spun edges

  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**reverse_faces(bm, faces=[], flip_multires=False)**

Reverse Faces.

Reverses the winding (vertex order) of faces. This has the effect of flipping the normal.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **flip_multires** (*bool*) – maintain multi-res offset

bmesh.ops.**flip_quad_tessellation(bm, faces=[])**

Flip Quad Tessellation

Flip the tessellation direction of the selected quads.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – Undocumented.

bmesh.ops.**bisect_edges(bm, edges=[], cuts=0, edge_percents={})**

Edge Bisect.

Splits input edges (but doesn't do anything else). This creates a 2-valence vert.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **cuts** (*int*) – number of cuts
- **edge_percents** (*dict mapping vert/edge/face types to float*) – Undocumented.

RETURNS:

- `geom_split` : newly created vertices and edges

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**mirror(bm, geom=[], matrix=mathutils.Matrix.Identity(4), merge_dist=0, axis='X', mirror_u=False, mirror_v=False, mirror_udim=False, use_shapekey=False)**

Mirror.

Mirrors geometry along an axis. The resulting geometry is welded on using merge_dist. Pairs of original/mirrored vertices are welded using the merge_dist parameter (which defines the minimum distance for welding to happen).

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **matrix** (`mathutils.Matrix`) – matrix defining the mirror transformation
- **merge_dist** (*float*) – maximum distance for merging. does no merging if 0.
- **axis** (*enum in ['X', 'Y', 'Z'], default 'X'*) – the axis to use.
- **mirror_u** (*bool*) – mirror UVs across the u axis
- **mirror_v** (*bool*) – mirror UVs across the v axis
- **mirror_udim** (*bool*) – mirror UVs in each tile
- **use_shapekey** (*bool*) – Transform shape keys too.

RETURNS:

- `geom` : output geometry, mirrored

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**find_doubles(bm, verts=[], keep_verts=[], dist=0)**

Find Doubles.

Takes input verts and find vertices they should weld to. Outputs a mapping slot suitable for use with the weld verts BMOP.

If keep_verts is used, vertices outside that set can only be merged with vertices in that set.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **keep_verts** (list of (`bmesh.types.BMVert`)) – list of verts to keep
- **dist** (*float*) – maximum distance

RETURNS:

- `targetmap`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert` / `bmesh.types.BMEdge` / `bmesh.types.BMFace`

RETURN TYPE:
dict[str, Any]

bmesh.ops.**remove_doubles(bm, verts=[], dist=0)**

Remove Doubles.

Finds groups of vertices closer than dist and merges them together, using the weld verts BMOP.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input verts
- **dist** (*float*) – minimum distance

bmesh.ops.**collapse(bm, edges=[], uvs=False)**

Collapse Connected.

Collapses connected vertices

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **uvs** (*bool*) – also collapse UVs and such

bmesh.ops.**pointmerge_facedata(bm, verts=[], vert_snap)**

Face-Data Point Merge.

Merge uv/vcols at a specific vertex.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **vert_snap** (`bmesh.types.BMVert`) – snap vertex

bmesh.ops.**average_vert_facedata(bm, verts=[])**

Average Vertices Face-vert Data.

Merge uv/vcols associated with the input vertices at the bounding box center. (I know, it's not averaging but the vert_snap_to_bb_center is just too long).

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **verts** (list of (bmesh.types.BMVert)) – input vertices

bmesh.ops.**pointmerge(bm, verts=[], merge_co=mathutils.Vector())**

Point Merge.

Merge verts together at a point.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **verts** (list of (bmesh.types.BMVert)) – input vertices (all verts will be merged into the first).
- **merge_co** (mathutils.Vector or any sequence of 3 floats) – Position to merge at.

bmesh.ops.**collapse_uvs(bm, edges=[])**

Collapse Connected UVs.

Collapses connected UV vertices.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **edges** (list of (bmesh.types.BMEdge)) – input edges

bmesh.ops.**weld_verts(bm, targetmap={})**

Weld Verts.

Welds verts together (kind-of like remove doubles, merge, etc, all of which use or will use this BMOP). You pass in mappings from vertices to the vertices they weld with.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **targetmap** (dict mapping vert/edge/face types to bmesh.types.BMVert/bmesh.types.BMEdge/bmesh.types.BMFace) – maps welded vertices to verts they should weld to

bmesh.ops.**create_vert(bm, co=mathutils.Vector())**

Make Vertex.

Creates a single vertex; this BMOP was necessary for click-create-vertex.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **co** (mathutils.Vector or any sequence of 3 floats) – the coordinate of the new vert

RETURNS:

- vert : the new vert

  type list of (bmesh.types.BMVert)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**join_triangles(bm, faces=[], cmp_seam=False, cmp_sharp=False, cmp_uvs=False, cmp_vcols=False, cmp_materials=False, angle_face_threshold=0, angle_shape_threshold=0, topology_influence=0, deselect_joined=False, merge_limit=0, neighbor_debug=0)**

Join Triangles.

Tries to intelligently join triangles according to angle threshold and delimiters.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.

- **faces** (list of (`bmesh.types.BMFace`)) – input geometry.
- **cmp_seam** (*bool*) – Compare seam
- **cmp_sharp** (*bool*) – Compare sharp
- **cmp_uvs** (*bool*) – Compare UVs
- **cmp_vcols** (*bool*) – compare VCols
- **cmp_materials** (*bool*) – compare materials
- **angle_face_threshold** (*float*) – Undocumented.
- **angle_shape_threshold** (*float*) – Undocumented.
- **topology_influence** (*float*) – Undocumented.
- **deselect_joined** (*bool*) – Undocumented.
- **merge_limit** (*int*) – Undocumented.
- **neighbor_debug** (*int*) – Undocumented.

RETURNS:

- `faces` : joined faces
  type list of (`bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**contextual_create(bm, geom=[], mat_nr=0, use_smooth=False)**

Contextual Create.

This is basically F-key, it creates new faces from vertices, makes stuff from edge nets, makes wire edges, etc. It also dissolves faces.

Three verts become a triangle, four become a quad. Two become a wire edge.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry.
- **mat_nr** (*int*) – material to use
- **use_smooth** (*bool*) – smooth to use

RETURNS:

- `faces` : newly-made face(s)
  type list of (`bmesh.types.BMFace`)
- `edges` : newly-made edge(s)
  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**bridge_loops(bm, edges=[], use_pairs=False, use_cyclic=False, use_merge=False, merge_factor=0, twist_offset=0)**

Bridge edge loops with faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **use_pairs** (*bool*) – Undocumented.
- **use_cyclic** (*bool*) – Undocumented.
- **use_merge** (*bool*) – merge rather than creating faces
- **merge_factor** (*float*) – merge factor
- **twist_offset** (*int*) – twist offset for closed loops

RETURNS:

- `faces` : new faces

  type list of (`bmesh.types.BMFace`)

- `edges` : new edges

  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**grid_fill(bm, edges=[], mat_nr=0, use_smooth=False, use_interp_simple=False)**

Grid Fill.

Create faces defined by 2 disconnected edge loops (which share edges).

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **mat_nr** (*int*) – material to use
- **use_smooth** (*bool*) – smooth state to use
- **use_interp_simple** (*bool*) – use simple interpolation

RETURNS:

- `faces` : new faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**holes_fill(bm, edges=[], sides=0)**

Fill Holes.

Fill boundary edges with faces, copying surrounding custom-data.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **sides** (*int*) – number of face sides to fill

RETURNS:

- `faces` : new faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**face_attribute_fill(bm, faces=[], use_normals=False, use_data=False)**

Face Attribute Fill.

Fill in faces with data from adjacent faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **use_normals** (*bool*) – copy face winding
- **use_data** (*bool*) – copy face data

RETURNS:

- `faces_fail` : faces that could not be handled

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**edgeloop_fill(bm, edges=[], mat_nr=0, use_smooth=False)**

Edge Loop Fill.

Create faces defined by one or more non overlapping edge loops.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **mat_nr** (*int*) – material to use
- **use_smooth** (*bool*) – smooth state to use

RETURNS:

- `faces` : new faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**edgenet_fill(bm, edges=[], mat_nr=0, use_smooth=False, sides=0)**

Edge Net Fill.

Create faces defined by enclosed edges.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **mat_nr** (*int*) – material to use
- **use_smooth** (*bool*) – smooth state to use
- **sides** (*int*) – number of sides

RETURNS:

- `faces` : new faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**edgenet_prepare(bm, edges=[])**

Edge-net Prepare.

Identifies several useful edge loop cases and modifies them so they'll become a face when edgenet_fill is called. The cases covered are:

- One single loop; an edge is added to connect the ends
- Two loops; two edges are added to connect the endpoints (based on the shortest distance between each endpoint).

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges

RETURNS:

- `edges` : new edges

  type list of (`bmesh.types.BMEdge`)

type list of (bmesh.types.BMEdge)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**rotate(bm, cent=mathutils.Vector(), matrix=mathutils.Matrix.Identity(4), verts=[], space=mathutils.Matrix.Identity(4), use_shapekey=False)**

Rotate.

Rotate vertices around a center, using a 3x3 rotation matrix.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **cent** (`mathutils.Vector` or any sequence of 3 floats) – center of rotation
- **matrix** (`mathutils.Matrix`) – matrix defining rotation
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **space** (`mathutils.Matrix`) – matrix to define the space (typically object matrix)
- **use_shapekey** (*bool*) – Transform shape keys too.

bmesh.ops.**translate(bm, vec=mathutils.Vector(), space=mathutils.Matrix.Identity(4), verts=[], use_shapekey=False)**

Translate.

Translate vertices by an offset.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **vec** (`mathutils.Vector` or any sequence of 3 floats) – translation offset
- **space** (`mathutils.Matrix`) – matrix to define the space (typically object matrix)
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **use_shapekey** (*bool*) – Transform shape keys too.

bmesh.ops.**scale(bm, vec=mathutils.Vector(), space=mathutils.Matrix.Identity(4), verts=[], use_shapekey=False)**

Scale.

Scales vertices by an offset.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **vec** (`mathutils.Vector` or any sequence of 3 floats) – scale factor
- **space** (`mathutils.Matrix`) – matrix to define the space (typically object matrix)
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **use_shapekey** (*bool*) – Transform shape keys too.

bmesh.ops.**transform(bm, matrix=mathutils.Matrix.Identity(4), space=mathutils.Matrix.Identity(4), verts=[], use_shapekey=False)**

Transform.

Transforms a set of vertices by a matrix. Multiplies the vertex coordinates with the matrix.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **matrix** (`mathutils.Matrix`) – transform matrix
- **space** (`mathutils.Matrix`) – matrix to define the space (typically object matrix)
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **use_shapekey** (*bool*) – Transform shape keys too.

bmesh.ops.**object_load_bmesh(bm, scene, object)**

Object Load BMesh.

Loads a bmesh into an object/mesh. This is a “private” BMOP.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **scene** (`bpy.types.Scene`) – pointer to an scene structure
- **object** (`bpy.types.Object`) – pointer to an object structure

bmesh.ops.**bmesh_to_mesh(bm, mesh, object)**

BMesh to Mesh.

Converts a bmesh to a Mesh. This is reserved for exiting editmode.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **mesh** (`bpy.types.Mesh`) – pointer to a mesh structure to fill in
- **object** (`bpy.types.Object`) – pointer to an object structure

bmesh.ops.**mesh_to_bmesh(bm, mesh, object, use_shapekey=False)**

Mesh to BMesh.

Load the contents of a mesh into the bmesh. this BMOP is private, it’s reserved exclusively for entering editmode.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **mesh** (`bpy.types.Mesh`) – pointer to a Mesh structure
- **object** (`bpy.types.Object`) – pointer to an Object structure
- **use_shapekey** (*bool*) – load active shapekey coordinates into verts

bmesh.ops.**extrude_discrete_faces(bm, faces=[], use_normal_flip=False, use_select_history=False)**

Individual Face Extrude.

Extrudes faces individually.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **use_normal_flip** (*bool*) – Create faces with reversed direction.
- **use_select_history** (*bool*) – pass to duplicate

RETURNS:

- `faces` : output faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**extrude_edge_only(bm, edges=[], use_normal_flip=False, use_select_history=False)**

Extrude Only Edges.

Extrudes Edges into faces, note that this is very simple, there’s no fancy winged extrusion.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input vertices
- **use_normal_flip** (*bool*) – Create faces with reversed direction.
- **use_select_history** (*bool*) – pass to duplicate

RETURNS:

- `geom` : output geometry

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**extrude_vert_indiv(bm, verts=[], use_select_history=False)**

Individual Vertex Extrude.

Extrudes wire edges from vertices.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **use_select_history** (*bool*) – pass to duplicate

RETURNS:

- `edges` : output wire edges

  type list of (`bmesh.types.BMEdge`)

- `verts` : output vertices

  type list of (`bmesh.types.BMVert`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**connect_verts(bm, verts=[], faces_exclude=[], check_degenerate=False)**

Connect Verts.

Split faces by adding edges that connect **verts**.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **faces_exclude** (list of (`bmesh.types.BMFace`)) – input faces to explicitly exclude from connecting
- **check_degenerate** (*bool*) – prevent splits with overlaps & intersections

RETURNS:

- `edges` :

  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**connect_verts_concave(bm, faces=[])**

Connect Verts to form Convex Faces.

Ensures all faces are convex **faces**.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces

RETURNS:

- `edges` :

  type list of (`bmesh.types.BMEdge`)

- `faces`:

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**connect_verts_nonplanar(bm, angle_limit=0, faces=[])**

Connect Verts Across non Planer Faces.

Split faces by connecting edges along non planer **faces**.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **angle_limit** (*float*) – total rotation angle (radians)
- **faces** (list of (`bmesh.types.BMFace`)) – input faces

RETURNS:

- `edges`:

  type list of (`bmesh.types.BMEdge`)

- `faces`:

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**connect_vert_pair(bm, verts=[], verts_exclude=[], faces_exclude=[])**

Connect Verts.

Split faces by adding edges that connect **verts**.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **verts_exclude** (list of (`bmesh.types.BMVert`)) – input vertices to explicitly exclude from connecting
- **faces_exclude** (list of (`bmesh.types.BMFace`)) – input faces to explicitly exclude from connecting

RETURNS:

- `edges`:

  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**extrude_face_region(bm, geom=[], edges_exclude=set(), use_keep_orig=False, use_normal_flip=False, use_normal_from_adjacent=False, use_dissolve_ortho_edges=False, use_select_history=False)**

Extrude Faces.

Extrude operator (does not transform)

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)) – edges and faces
- **edges_exclude** (*set of vert/edge/face type*) – input edges to explicitly exclude from extrusion
- **use_keep_orig** (*bool*) – keep original geometry (requires `geom` to include edges).
- **use_normal_flip** (*bool*) – Create faces with reversed direction.
- **use_normal_from_adjacent** (*bool*) – Use winding from surrounding faces instead of this region.
- **use_dissolve_ortho_edges** (*bool*) – Dissolve edges whose faces form a flat surface

- **use_dissolve_ortho_edges** (*bool*) – Dissolve edges whose faces form a flat surface.
- **use_select_history** (*bool*) – pass to duplicate

RETURNS:

- `geom`:
  type list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**dissolve_verts(bm, verts=[], use_face_split=False, use_boundary_tear=False)**

Dissolve Verts.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **use_face_split** (*bool*) – split off face corners to maintain surrounding geometry
- **use_boundary_tear** (*bool*) – split off face corners instead of merging faces

bmesh.ops.**dissolve_edges(bm, edges=[], use_verts=False, use_face_split=False)**

Dissolve Edges.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **use_verts** (*bool*) – dissolve verts left between only 2 edges.
- **use_face_split** (*bool*) – split off face corners to maintain surrounding geometry

RETURNS:

- `region`:
  type list of (`bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**dissolve_faces(bm, faces=[], use_verts=False)**

Dissolve Faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **use_verts** (*bool*) – dissolve verts left between only 2 edges.

RETURNS:

- `region`:
  type list of (`bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**dissolve_limit(bm, angle_limit=0, use_dissolve_boundaries=False, verts=[], edges=[], delimit=set())**

Limited Dissolve.

Dissolve planar faces and co-linear edges.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.

- **angle_limit** (*float*) – total rotation angle (radians)
- **use_dissolve_boundaries** (*bool*) – dissolve all vertices in between face boundaries
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **delimit** (*set of flags from ['NORMAL', 'MATERIAL', 'SEAM', 'SHARP', 'UV'], default set()*) – delimit dissolve operation

RETURNS:

- `region` :
  type list of (`bmesh.types.BMFace`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**dissolve_degenerate(bm, dist=0, edges=[])**

Degenerate Dissolve.

Dissolve edges with no length, faces with no area.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **dist** (*float*) – maximum distance to consider degenerate
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges

bmesh.ops.**triangulate(bm, faces=[], quad_method='BEAUTY', ngon_method='BEAUTY')**

Triangulate.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **quad_method** (*enum in ['BEAUTY', 'FIXED', 'ALTERNATE', 'SHORT_EDGE', 'LONG_EDGE'], default 'BEAUTY'*) – method for splitting the quads into triangles
- **ngon_method** (*enum in ['BEAUTY', 'EAR_CLIP'], default 'BEAUTY'*) – method for splitting the polygons into triangles

RETURNS:

- `edges` :
  type list of (`bmesh.types.BMEdge`)
- `faces` :
  type list of (`bmesh.types.BMFace`)
- `face_map` :
  type dict mapping vert/edge/face types to `bmesh.types.BMVert` / `bmesh.types.BMEdge` / `bmesh.types.BMFace`
- `face_map_double` : duplicate faces
  type dict mapping vert/edge/face types to `bmesh.types.BMVert` / `bmesh.types.BMEdge` / `bmesh.types.BMFace`

RETURN TYPE:
dict[str, Any]

bmesh.ops.**unsubdivide(bm, verts=[], iterations=0)**

Un-Subdivide.

Reduce detail in geometry containing grids.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **verts** (list of (`bmesh.types.BMVert`)) – input vertices

- **iterations** (*int*) – number of times to unsubdivide

bmesh.ops.**subdivide_edges(bm, edges=[], smooth=0, smooth_falloff='SMOOTH', fractal=0, along_normal=0, cuts=0, seed=0, custom_patterns={}, edge_percents={}, quad_corner_type='STRAIGHT_CUT', use_grid_fill=False, use_single_edge=False, use_only_quads=False, use_sphere=False, use_smooth_even=False)**

Subdivide Edges.

Advanced operator for subdividing edges with options for face patterns, smoothing and randomization.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **smooth** (*float*) – smoothness factor
- **smooth_falloff** (*enum in ['SMOOTH', 'SPHERE', 'ROOT', 'SHARP', 'LINEAR', 'INVERSE_SQUARE'], default 'SMOOTH'*) – smooth falloff type
- **fractal** (*float*) – fractal randomness factor
- **along_normal** (*float*) – apply fractal displacement along normal only
- **cuts** (*int*) – number of cuts
- **seed** (*int*) – seed for the random number generator
- **custom_patterns** (*dict mapping vert/edge/face types to unknown internal data, not compatible with python*) – uses custom pointers
- **edge_percents** (*dict mapping vert/edge/face types to float*) – Undocumented.
- **quad_corner_type** (*enum in ['STRAIGHT_CUT', 'INNER_VERT', 'PATH', 'FAN'], default 'STRAIGHT_CUT'*) – quad corner type
- **use_grid_fill** (*bool*) – fill in fully-selected faces with a grid
- **use_single_edge** (*bool*) – tessellate the case of one edge selected in a quad or triangle
- **use_only_quads** (*bool*) – Only subdivide quads (for loop-cut).
- **use_sphere** (*bool*) – for making new primitives only
- **use_smooth_even** (*bool*) – maintain even offset when smoothing

RETURNS:

- `geom_inner`:

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

- `geom_split`:

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

- `geom` : contains all output geometry

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**subdivide_edgering(bm, edges=[], interp_mode='LINEAR', smooth=0, cuts=0, profile_shape='SMOOTH', profile_shape_factor=0)**

Subdivide Edge-Ring.

Take an edge-ring, and subdivide with interpolation options.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input vertices
- **interp_mode** (*enum in ['LINEAR', 'PATH', 'SURFACE'], default 'LINEAR'*) – interpolation method
- **smooth** (*float*) – smoothness factor
- **cuts** (*int*) – number of cuts
- **profile_shape** (*enum in ['SMOOTH', 'SPHERE', 'ROOT', 'SHARP', 'LINEAR', 'INVERSE_SQUARE'], default 'SMOOTH'*) – profile sha type

- **profile_shape_factor** (*float*) – how much intermediary new edges are shrunk/expanded

RETURNS:

- `faces` : output faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**bisect_plane(bm, geom=[], dist=0, plane_co=mathutils.Vector(), plane_no=mathutils.Vector(), use_snap_center=False, clear_outer=False, clear_inner=False)**

Bisect Plane.

Bisects the mesh by a plane (cut the mesh in half).

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **dist** (*float*) – minimum distance when testing if a vert is exactly on the plane
- **plane_co** (`mathutils.Vector` or any sequence of 3 floats) – point on the plane
- **plane_no** (`mathutils.Vector` or any sequence of 3 floats) – direction of the plane
- **use_snap_center** (*bool*) – snap axis aligned verts to the center
- **clear_outer** (*bool*) – when enabled. remove all geometry on the positive side of the plane
- **clear_inner** (*bool*) – when enabled. remove all geometry on the negative side of the plane

RETURNS:

- `geom_cut` : output geometry aligned with the plane (new and existing)

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge`)

- `geom` : input and output geometry (result of cut).

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**delete(bm, geom=[], context='VERTS')**

Delete Geometry.

Utility operator to delete geometry.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **context** (*enum in ['VERTS', 'EDGES', 'FACES_ONLY', 'EDGES_FACES', 'FACES', 'FACES_KEEP_BOUNDARY', 'TAGGED_ONLY'], default 'VERTS'*) – geometry types to delete

bmesh.ops.**duplicate(bm, geom=[], dest=None, use_select_history=False, use_edge_flip_from_face=False)**

Duplicate Geometry.

Utility operator to duplicate geometry, optionally into a destination mesh.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **dest** (`bmesh.types.BMesh`) – destination bmesh, if None will use current on
- **use_select_history** (*bool*) – Undocumented.
- **use_edge_flip_from_face** (*bool*) – Undocumented.

RETURNS:

- `geom_orig`:
  type list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)
- `geom`:
  type list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)
- `vert_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`
- `edge_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`
- `face_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`
- `boundary_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`
- `isovert_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`

RETURN TYPE:
dict[str, Any]

bmesh.ops.**split(bm, geom=[], dest=None, use_only_faces=False)**

Split Off Geometry.

Disconnect geometry from adjacent edges and faces, optionally into a destination mesh.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)) – input geometry
- **dest** (`bmesh.types.BMesh`) – destination bmesh, if None will use current one
- **use_only_faces** (*bool*) – when enabled. don't duplicate loose verts/edges

RETURNS:

- `geom`:
  type list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)
- `boundary_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`
- `isovert_map`:
  type dict mapping vert/edge/face types to `bmesh.types.BMVert`/`bmesh.types.BMEdge`/`bmesh.types.BMFace`

RETURN TYPE:
dict[str, Any]

bmesh.ops.**spin(bm, geom=[], cent=mathutils.Vector(), axis=mathutils.Vector(), dvec=mathutils.Vector(), angle=0, space=mathutils.Matrix.Identity(4), steps=0, use_merge=False, use_normal_flip=False, use_duplicate=False)**

Spin.

Extrude or duplicate geometry a number of times, rotating and possibly translating after each step

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **geom** (list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)) – input geometry
- **cent** (`mathutils.Vector` or any sequence of 3 floats) – rotation center

- **axis** (`mathutils.Vector` or any sequence of 3 floats) – rotation axis
- **dvec** (`mathutils.Vector` or any sequence of 3 floats) – translation delta per step
- **angle** (*float*) – total rotation angle (radians)
- **space** (`mathutils.Matrix`) – matrix to define the space (typically object matrix)
- **steps** (*int*) – number of steps
- **use_merge** (*bool*) – Merge first/last when the angle is a full revolution.
- **use_normal_flip** (*bool*) – Create faces with reversed direction.
- **use_duplicate** (*bool*) – duplicate or extrude?

RETURNS:

- `geom_last` : result of last step

  type list of (`bmesh.types.BMVert`, `bmesh.types.BMEdge`, `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**rotate_uvs(bm, faces=[], use_ccw=False)**

UV Rotation.

Cycle the loop UVs

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **use_ccw** (*bool*) – rotate counter-clockwise if true, otherwise clockwise

bmesh.ops.**reverse_uvs(bm, faces=[])**

UV Reverse.

Reverse the UVs

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces

bmesh.ops.**rotate_colors(bm, faces=[], use_ccw=False, color_index=0)**

Color Rotation.

Cycle the loop colors

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **use_ccw** (*bool*) – rotate counter-clockwise if true, otherwise clockwise
- **color_index** (*int*) – index into color attribute list

bmesh.ops.**reverse_colors(bm, faces=[], color_index=0)**

Color Reverse

Reverse the loop colors.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **color_index** (*int*) – index into color attribute list

bmesh.ops.**split_edges(bm, edges=[], verts=[], use_verts=False)**

Edge Split.

Disconnects faces along input edges.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **verts** (list of (`bmesh.types.BMVert`)) – optional tag verts, use to have greater control of splits
- **use_verts** (*bool*) – use 'verts' for splitting, else just find verts to split from edges

RETURNS:

- `edges` : old output disconnected edges

  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**create_grid(bm, x_segments=0, y_segments=0, size=0, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Create Grid.

Creates a grid with a variable number of subdivisions

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **x_segments** (*int*) – number of x segments
- **y_segments** (*int*) – number of y segments
- **size** (*float*) – size of the grid
- **matrix** (`mathutils.Matrix`) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**create_uvsphere(bm, u_segments=0, v_segments=0, radius=0, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Create UV Sphere.

Creates a grid with a variable number of subdivisions

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **u_segments** (*int*) – number of u segments
- **v_segments** (*int*) – number of v segment
- **radius** (*float*) – radius
- **matrix** (`mathutils.Matrix`) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**create_icosphere(bm, subdivisions=0, radius=0, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Create Ico-Sphere.

Creates a grid with a variable number of subdivisions

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **subdivisions** (*int*) – how many times to recursively subdivide the sphere
- **radius** (*float*) – radius
- **matrix** (`mathutils.Matrix`) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**create_monkey(bm, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Create Suzanne.

Creates a monkey (standard blender primitive).

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **matrix** (`mathutils.Matrix`) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**create_cone(bm, cap_ends=False, cap_tris=False, segments=0, radius1=0, radius2=0, depth=0, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Create Cone.

Creates a cone with variable depth at both ends

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **cap_ends** (*bool*) – whether or not to fill in the ends with faces
- **cap_tris** (*bool*) – fill ends with triangles instead of ngons
- **segments** (*int*) – number of vertices in the base circle
- **radius1** (*float*) – radius of one end
- **radius2** (*float*) – radius of the opposite
- **depth** (*float*) – distance between ends
- **matrix** (`mathutils.Matrix`) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

type list of (bmesh.types.BMVert)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**create_circle(bm, cap_ends=False, cap_tris=False, segments=0, radius=0, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Creates a Circle.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **cap_ends** (*bool*) – whether or not to fill in the ends with faces
- **cap_tris** (*bool*) – fill ends with triangles instead of ngons
- **segments** (*int*) – number of vertices in the circle
- **radius** (*float*) – Radius of the circle.
- **matrix** (mathutils.Matrix) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- verts : output verts
  type list of (bmesh.types.BMVert)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**create_cube(bm, size=0, matrix=mathutils.Matrix.Identity(4), calc_uvs=False)**

Create Cube

Creates a cube.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **size** (*float*) – size of the cube
- **matrix** (mathutils.Matrix) – matrix to multiply the new geometry with
- **calc_uvs** (*bool*) – calculate default UVs

RETURNS:

- verts : output verts
  type list of (bmesh.types.BMVert)

RETURN TYPE:
dict[str, Any]

bmesh.ops.**bevel(bm, geom=[], offset=0, offset_type='OFFSET', profile_type='SUPERELLIPSE', segments=0, profile=0, affect='VERTICES', clamp_overlap=False, material=0, loop_slide=False, mark_seam=False, mark_sharp=False, harden_normals=False, face_strength_mode='NONE', miter_outer='SHARP', miter_inner='SHARP', spread=0, custom_profile=None, vmesh_method='ADJ')**

Bevel.

Bevels edges and vertices

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **geom** (list of (bmesh.types.BMVert , bmesh.types.BMEdge , bmesh.types.BMFace)) – input edges and vertices
- **offset** (*float*) – amount to offset beveled edge
- **offset_type** (*enum in ['OFFSET', 'WIDTH', 'DEPTH', 'PERCENT', 'ABSOLUTE'], default 'OFFSET'*) – how to measure the offset
- **profile_type** (*enum in ['SUPERELLIPSE', 'CUSTOM'], default 'SUPERELLIPSE'*) – The profile type to use for bevel.

- **segments** (*int*) – number of segments in bevel
- **profile** (*float*) – profile shape, 0->1 (.5=>round)
- **affect** (*enum in ['VERTICES', 'EDGES'], default 'VERTICES'*) – Whether to bevel vertices or edges.
- **clamp_overlap** (*bool*) – do not allow beveled edges/vertices to overlap each other
- **material** (*int*) – material for bevel faces, -1 means get from adjacent faces
- **loop_slide** (*bool*) – prefer to slide along edges to having even widths
- **mark_seam** (*bool*) – extend edge data to allow seams to run across bevels
- **mark_sharp** (*bool*) – extend edge data to allow sharp edges to run across bevels
- **harden_normals** (*bool*) – harden normals
- **face_strength_mode** (*enum in ['NONE', 'NEW', 'AFFECTED', 'ALL'], default 'NONE'*) – whether to set face strength, and which faces t set if so
- **miter_outer** (*enum in ['SHARP', 'PATCH', 'ARC'], default 'SHARP'*) – outer miter kind
- **miter_inner** (*enum in ['SHARP', 'PATCH', 'ARC'], default 'SHARP'*) – outer miter kind
- **spread** (*float*) – amount to offset beveled edge
- **custom_profile** (`bpy.types.bpy_struct`) – CurveProfile, if None ignored
- **vmesh_method** (*enum in ['ADJ', 'CUTOFF'], default 'ADJ'*) – The method to use to create meshes at intersections.

RETURNS:

- `faces` : output faces

  type list of (`bmesh.types.BMFace`)
- `edges` : output edges

  type list of (`bmesh.types.BMEdge`)
- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**beautify_fill(bm, faces=[], edges=[], use_restrict_tag=False, method='AREA')**

Beautify Fill.

Rotate edges to create more evenly spaced triangles.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **edges** (list of (`bmesh.types.BMEdge`)) – edges that can be flipped
- **use_restrict_tag** (*bool*) – restrict edge rotation to mixed tagged vertices
- **method** (*enum in ['AREA', 'ANGLE'], default 'AREA'*) – method to define what is beautiful

RETURNS:

- `geom` : new flipped faces and edges

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**triangle_fill(bm, use_beauty=False, use_dissolve=False, edges=[], normal=mathutils.Vector())**

Triangle Fill.

Fill edges with triangles

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **use_beauty** (*bool*) – use best triangulation division
- **use_dissolve** (*bool*) – dissolve resulting faces
- **edges** (list of (bmesh.types.BMEdge)) – input edges
- **normal** (mathutils.Vector or any sequence of 3 floats) – optionally pass the fill normal to use

RETURNS:

- geom: new faces and edges

  type list of (bmesh.types.BMVert, bmesh.types.BMEdge, bmesh.types.BMFace)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**solidify(bm, geom=[], thickness=0)**

Solidify.

Turns a mesh into a shell with thickness

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **geom** (list of (bmesh.types.BMVert, bmesh.types.BMEdge, bmesh.types.BMFace)) – input geometry
- **thickness** (*float*) – thickness

RETURNS:

- geom:

  type list of (bmesh.types.BMVert, bmesh.types.BMEdge, bmesh.types.BMFace)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**inset_individual(bm, faces=[], thickness=0, depth=0, use_even_offset=False, use_interpolate=False, use_relative_offset=False)**

Face Inset (Individual).

Insets individual faces.

PARAMETERS:

- **bm** (bmesh.types.BMesh) – The bmesh to operate on.
- **faces** (list of (bmesh.types.BMFace)) – input faces
- **thickness** (*float*) – thickness
- **depth** (*float*) – depth
- **use_even_offset** (*bool*) – scale the offset to give more even thickness
- **use_interpolate** (*bool*) – blend face data across the inset
- **use_relative_offset** (*bool*) – scale the offset by surrounding geometry

RETURNS:

- faces: output faces

  type list of (bmesh.types.BMFace)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**inset_region(bm, faces=[], faces_exclude=[], use_boundary=False, use_even_offset=False, use_interpolate=False, use_relative_offset=False, use_edge_rail=False, thickness=0, depth=0, use_outset=False)**

Face Inset (Regions).

Inset or outset face regions.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **faces_exclude** (list of (`bmesh.types.BMFace`)) – input faces to explicitly exclude from inset
- **use_boundary** (*bool*) – inset face boundaries
- **use_even_offset** (*bool*) – scale the offset to give more even thickness
- **use_interpolate** (*bool*) – blend face data across the inset
- **use_relative_offset** (*bool*) – scale the offset by surrounding geometry
- **use_edge_rail** (*bool*) – inset the region along existing edges
- **thickness** (*float*) – thickness
- **depth** (*float*) – depth
- **use_outset** (*bool*) – outset rather than inset

RETURNS:

- `faces` : output faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**offset_edgeloops(bm, edges=[], use_cap_endpoint=False)**

Edge-loop Offset.

Creates edge loops based on simple edge-outset method.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **edges** (list of (`bmesh.types.BMEdge`)) – input edges
- **use_cap_endpoint** (*bool*) – extend loop around end-points

RETURNS:

- `edges` : output edges

  type list of (`bmesh.types.BMEdge`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**wireframe(bm, faces=[], thickness=0, offset=0, use_replace=False, use_boundary=False, use_even_offset=False, use_crease=False, crease_weight=0, use_relative_offset=False, material_offset=0)**

Wire Frame.

Makes a wire-frame copy of faces.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **thickness** (*float*) – thickness
- **offset** (*float*) – offset the thickness from the center
- **use_replace** (*bool*) – remove original geometry
- **use_boundary** (*bool*) – inset face boundaries
- **use_even_offset** (*bool*) – scale the offset to give more even thickness
- **use_crease** (*bool*) – crease hub edges for improved subdivision surface
- **crease_weight** (*float*) – the mean crease weight for resulting edges
- **use_relative_offset** (*bool*) – scale the offset by surrounding geometry

- **material_offset** (*int*) – offset material index of generated faces

RETURNS:

- `faces` : output faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**poke(bm, faces=[], offset=0, center_mode='MEAN_WEIGHTED', use_relative_offset=False)**

Pokes a face.

Splits a face into a triangle fan.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **faces** (list of (`bmesh.types.BMFace`)) – input faces
- **offset** (*float*) – center vertex offset along normal
- **center_mode** (*enum in ['MEAN_WEIGHTED', 'MEAN', 'BOUNDS'], default 'MEAN_WEIGHTED'*) – calculation mode for center verte
- **use_relative_offset** (*bool*) – apply offset

RETURNS:

- `verts` : output verts

  type list of (`bmesh.types.BMVert`)

- `faces` : output faces

  type list of (`bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**convex_hull(bm, input=[], use_existing_faces=False)**

Convex Hull

Builds a convex hull from the vertices in 'input'.

If 'use_existing_faces' is true, the hull will not output triangles that are covered by a pre-existing face.

All hull vertices, faces, and edges are added to 'geom.out'. Any input elements that end up inside the hull (i.e. are not used by an output face) are added to the 'interior_geom' slot. The 'unused_geom' slot will contain all interior geometry that is completely unused. Lastly, 'holes_geom' contain edges and faces that were in the input and are part of the hull.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **input** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **use_existing_faces** (*bool*) – skip hull triangles that are covered by a pre-existing face

RETURNS:

- `geom` :

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

- `geom_interior` :

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

- `geom_unused` :

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

- `geom_holes` :

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]

bmesh.ops.**symmetrize(bm, input=[], direction='-X', dist=0, use_shapekey=False)**

Symmetrize.

Makes the mesh elements in the "input" slot symmetrical. Unlike normal mirroring, it only copies in one direction, as specified by the "direction" slot The edges and faces that cross the plane of symmetry are split as needed to enforce symmetry.

All new vertices, edges, and faces are added to the "geom.out" slot.

PARAMETERS:

- **bm** (`bmesh.types.BMesh`) – The bmesh to operate on.
- **input** (list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)) – input geometry
- **direction** (*enum in ['-X', '-Y', '-Z', 'X', 'Y', 'Z'], default '-X'*) – axis to use
- **dist** (*float*) – minimum distance
- **use_shapekey** (*bool*) – Transform shape keys too.

RETURNS:

- `geom` :

  type list of (`bmesh.types.BMVert` , `bmesh.types.BMEdge` , `bmesh.types.BMFace`)

RETURN TYPE:

dict[str, Any]