



# Blender - Python API

#5



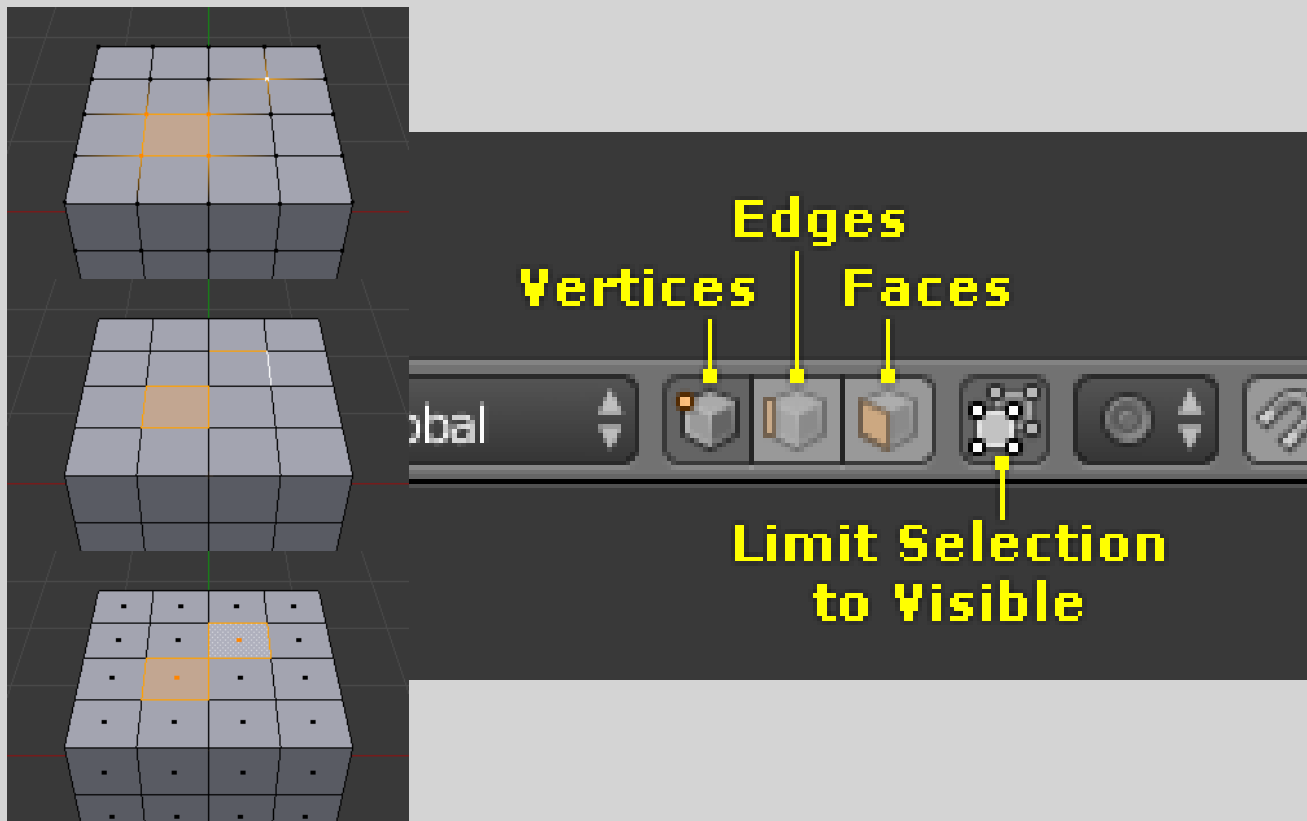
Serdar ARITAN

Department of Computer Graphics  
Hacettepe University, Ankara, Turkey



## Blender/Python API

### Selecting Vertices, Edges, and Faces by Location





### Selecting Vertices, Edges, and Faces by Location

**# Add in body of script, outside any class declarations**

```
def in_bbox(lbound, ubound, v, buffer=0.0001):
```

```
    return lbound[0] - buffer <= v[0] <= ubound[0] + buffer and \
           lbound[1] - buffer <= v[1] <= ubound[1] + buffer and \
           lbound[2] - buffer <= v[2] <= ubound[2] + buffer
```

```
class act:
```

```
    def select_by_loc(lbound=(0, 0, 0), ubound=(0, 0, 0),
                      select_mode='VERT', coords='GLOBAL'):
```

```
        # Set selection mode, VERT, EDGE, or FACE
        selection_mode(select_mode)
```

```
        # Grab the transformation matrix
        world = bpy.context.object.matrix_world
```



### Selecting Vertices, Edges, and Faces by Location

```
# Instantiate a bmesh object and ensure lookup table
# Running bm.faces.ensure_lookup_table() works for all parts

bm = bmesh.from_edit_mesh(bpy.context.object.data)
bm.faces.ensure_lookup_table()

# Initialize list of vertices and list of parts to be selected
verts = []
to_select = []

# For VERT, EDGE, or FACE ...
# 1. Grab list of global or local coordinates
# 2. Test if the piece is entirely within the rectangular
#    prism defined by lbound and ubound
# 3. Select each piece that returned True and deselect
#    each piece that returned False in Step 2
```



### Selecting Vertices, Edges, and Faces by Location

```
if select_mode == 'VERT':
    if coords == 'GLOBAL':
        [verts.append((world @ v.co).to_tuple()) for v in bm.verts]
    elif coords == 'LOCAL':
        [verts.append(v.co.to_tuple()) for v in bm.verts]

    [to_select.append(in_bbox(lbound, ubound, v)) for v in verts]
    for vertObj, select in zip(bm.verts, to_select):
        vertObj.select = select
if select_mode == 'EDGE':
    if coords == 'GLOBAL':
        [verts.append([(world @ v.co).to_tuple()
                        for v in e.verts]) for e in bm.edges]
    elif coords == 'LOCAL':
        [verts.append([v.co.to_tuple() for v in e.verts])
         for e in bm.edges]
    [to_select.append(all(in_bbox(lbound, ubound, v)
                          for v in e)) for e in verts]
    for edgeObj, select in zip(bm.edges, to_select):
        edgeObj.select = select
```



### Selecting Vertices, Edges, and Faces by Location

```
if select_mode == 'FACE':
    if coords == 'GLOBAL':
        [verts.append([(world @ v.co).to_tuple()
                        for v in f.verts]) for f in bm.faces]

    elif coords == 'LOCAL':
        [verts.append([v.co.to_tuple() for v in f.verts])
         for f in bm.faces]
        [to_select.append(all(in_bbox(lbound, ubound, v)
                                for v in f)) for f in verts]

for faceObj, select in zip(bm.faces, to_select):
    faceObj.select = select
```

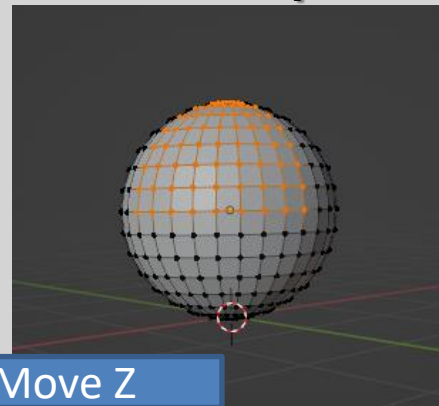


### Selecting and Transforming Pieces of a Sphere

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
# Move the sphere in Z direction
bpy.ops.transform.translate(value = (0, 0, 1))
tk.mode("EDIT")

# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'FACE', 'LOCAL')
```



Move Z



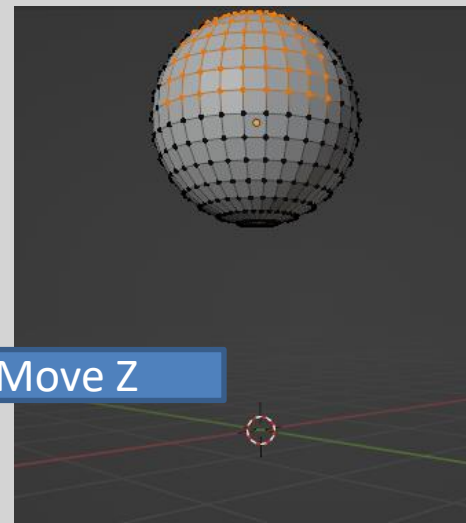


### Selecting and Transforming Pieces of a Sphere

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
# Move the sphere in Z direction
bpy.ops.transform.translate(value = (0, 0, 3))
tk.mode("EDIT")

# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'FACE', 'LOCAL')
```







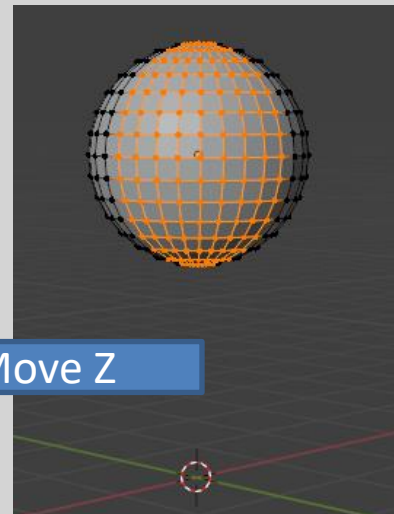
## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

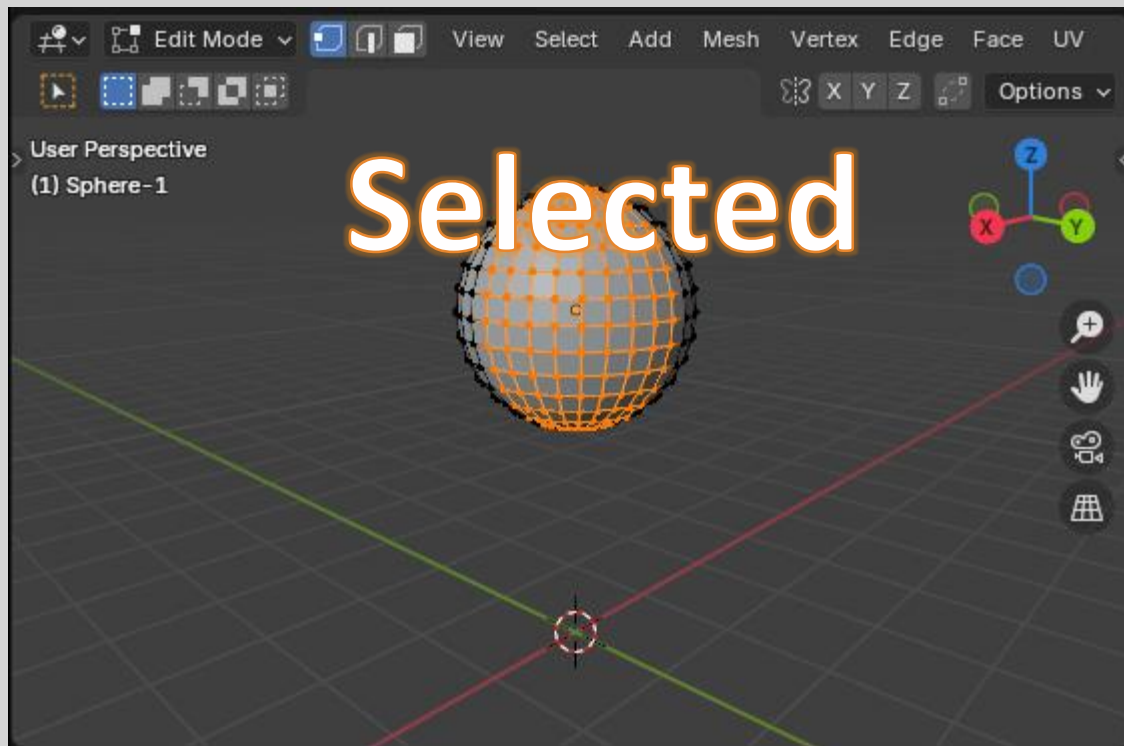
# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
# Move the sphere in Z direction
bpy.ops.transform.translate(value = (0, 0, 3))
tk.mode("EDIT")

# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, -1), (1, 1, 1), 'FACE', 'LOCAL')
```



# Blender/Python API

## Selecting and Transforming Pieces of a Sphere

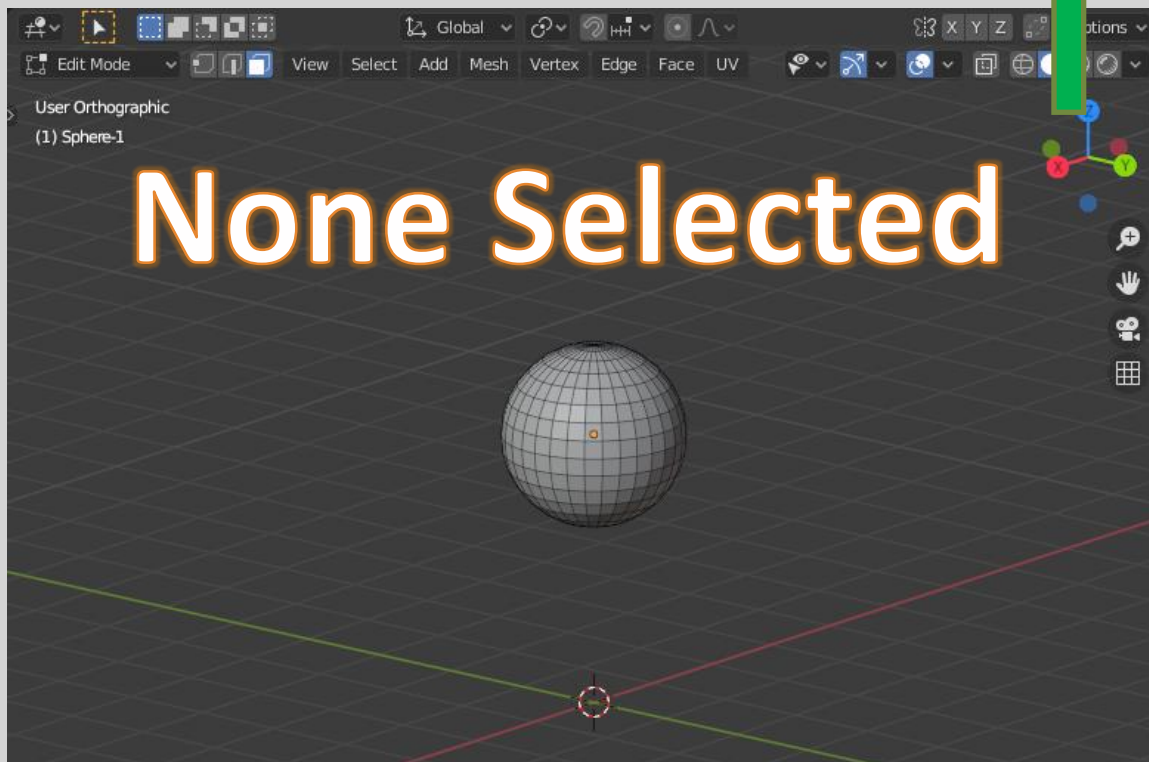




## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

```
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'FACE', 'GLOBAL')
```

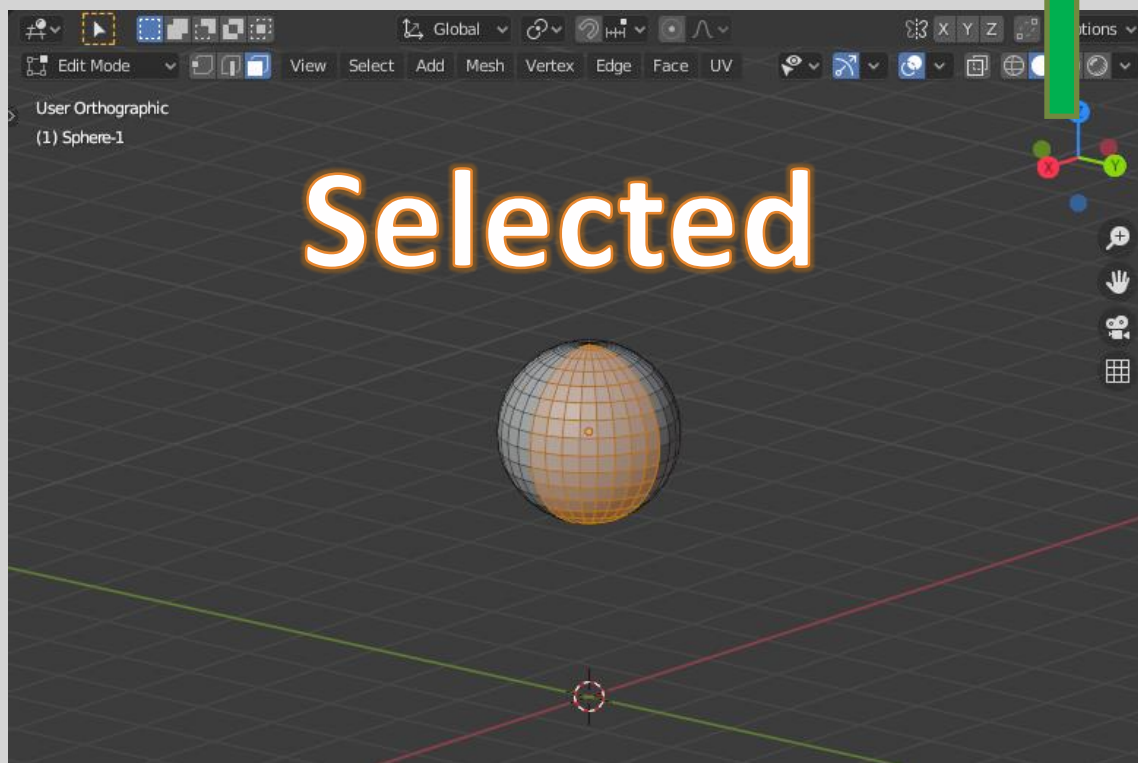




## Blender/Python API Selecting and Transforming Pieces of a Sphere



```
tk.act.select_by_loc((0, 0, 0), (5, 5, 5), 'FACE', 'GLOBAL')
```

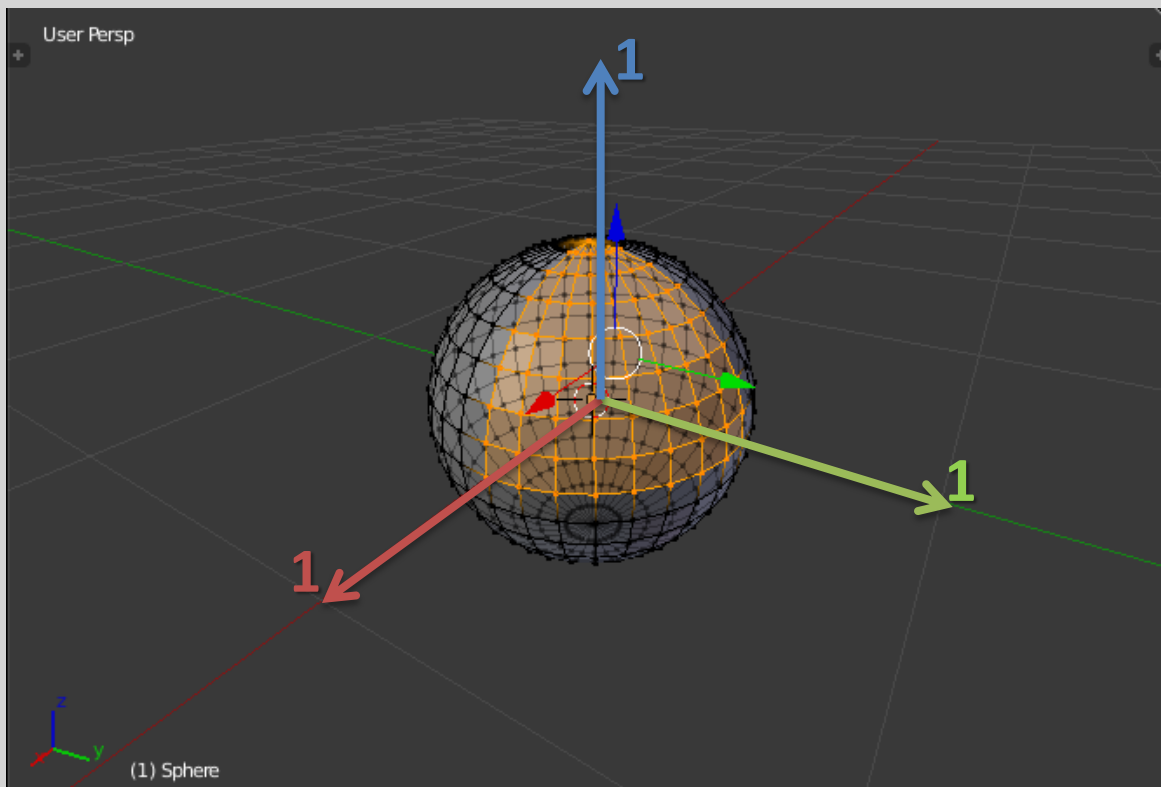




## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

# Selects upper right quadrant of sphere





## Blender/Python API

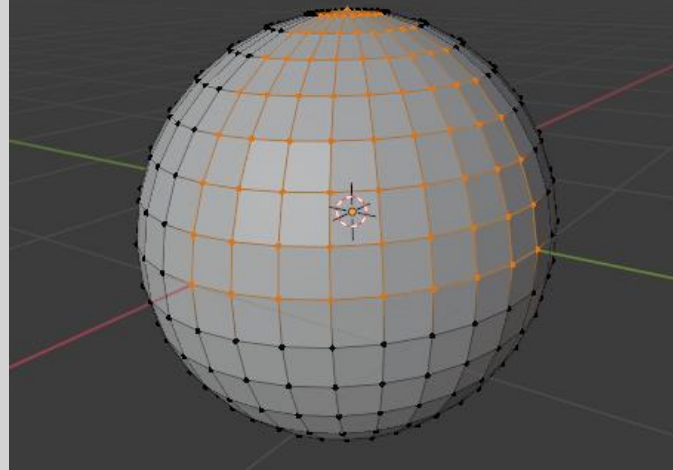
### Selecting and Transforming Pieces of a Sphere

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1') ← radius = 1
tk.mode("EDIT")
# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'VERT', 'LOCAL')
```

```
tk.act.select_by_loc((0, 0, 0), (1, 1, 1),  
                    'VERT', 'LOCAL')
```

# Selected







## Blender/Python API Selecting and Transforming Pieces of a Sphere

Create Sphere

Resize

3x

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()
```

```
# Will fail if scene is empty
```

```
tk.mode("OBJECT")
```

```
tk.delete_all()
```

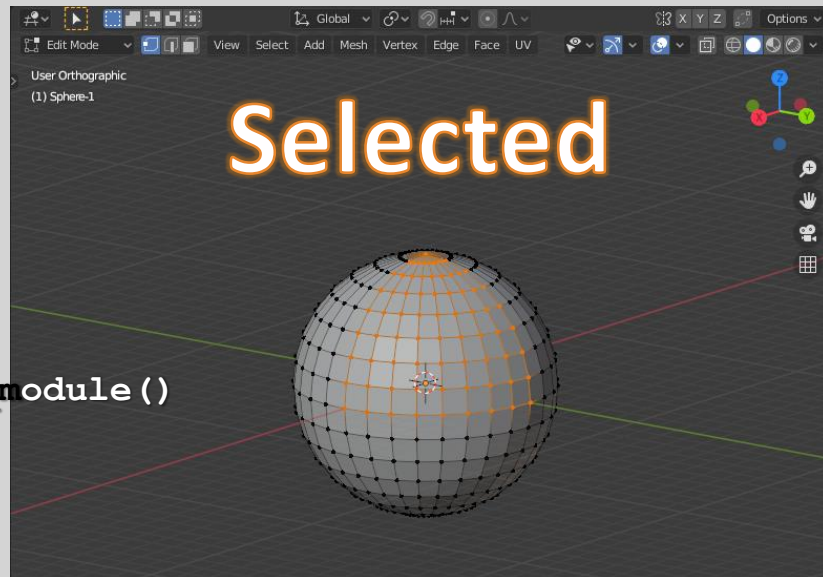
```
tk.create_sphere('Sphere-1') ← radius = 1
```

```
tk.sel.scale((5,)*3) ← resize # Scaling means changing proportions of objects
```

```
tk.mode("EDIT")
```

```
# Selects upper right quadrant of sphere
```

```
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'VERT', 'LOCAL')
```





### Selecting and Transforming Pieces of a Sphere

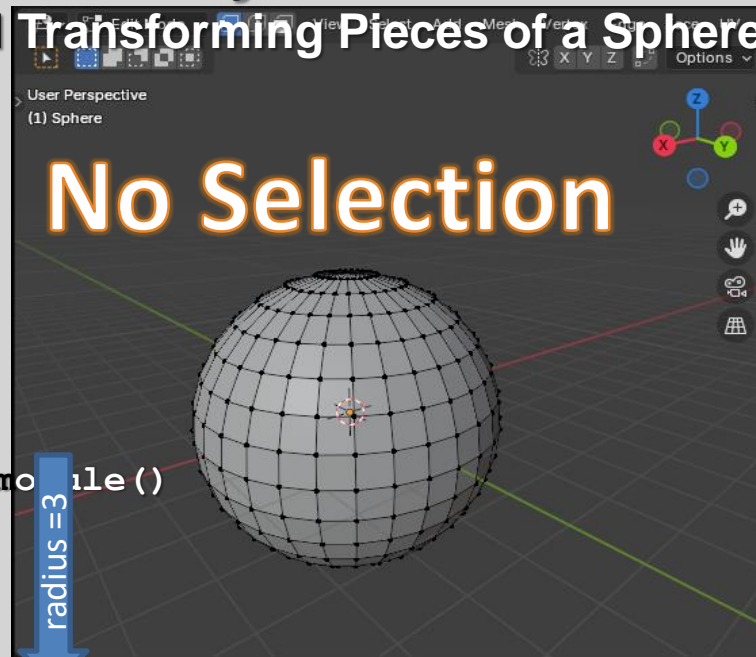
# No Resize



radius = 3

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_model()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
bpy.ops.mesh.primitive_uv_sphere_add(radius=3, location=(0, 0, 0), scale=(1, 1, 1))
tk.mode("EDIT")
# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'VERT', 'LOCAL')
```





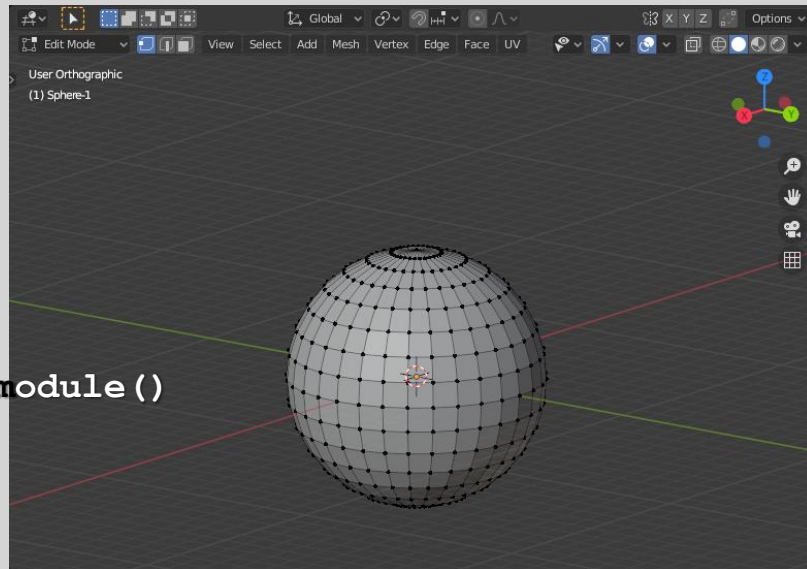


## Blender/Python API Selecting and Transforming Pieces of a Sphere



```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
tk.sel.scale((5,)*3)
tk.mode("EDIT")
# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'VERT', 'GLOBAL')
```



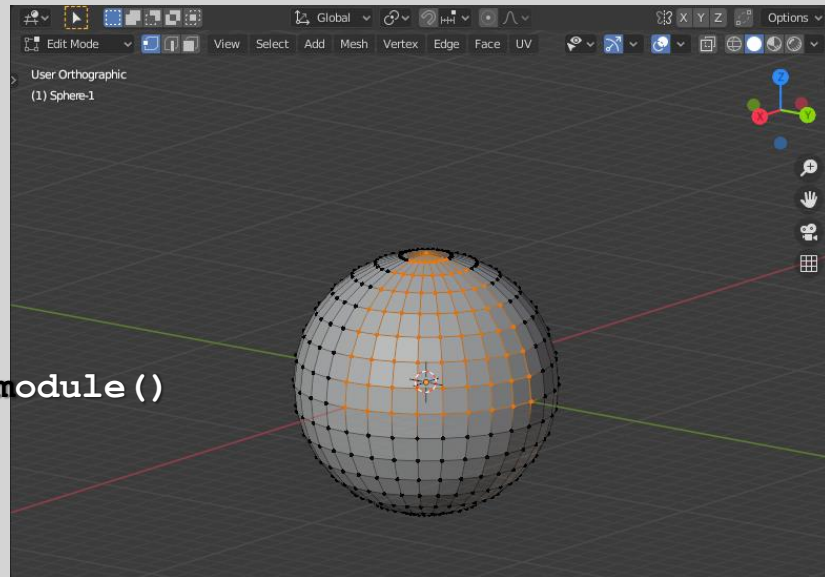


## Blender/Python API Selecting and Transforming Pieces of a Sphere



```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
tk.sel.scale((5,)*3)
tk.mode("EDIT")
# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (5, 5, 5), 'VERT', 'GLOBAL')
```



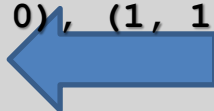
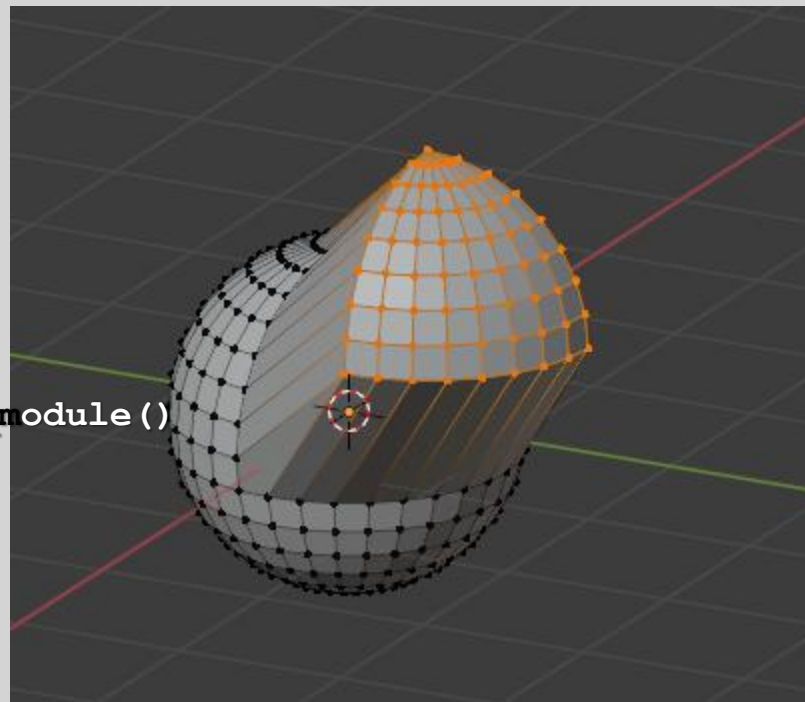


## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

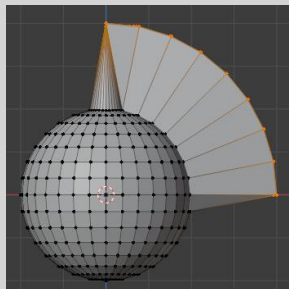
# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
tk.mode("EDIT")
# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'VERT', 'LOCAL')
tk.sel.translate((1, 1, 1))
```





## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

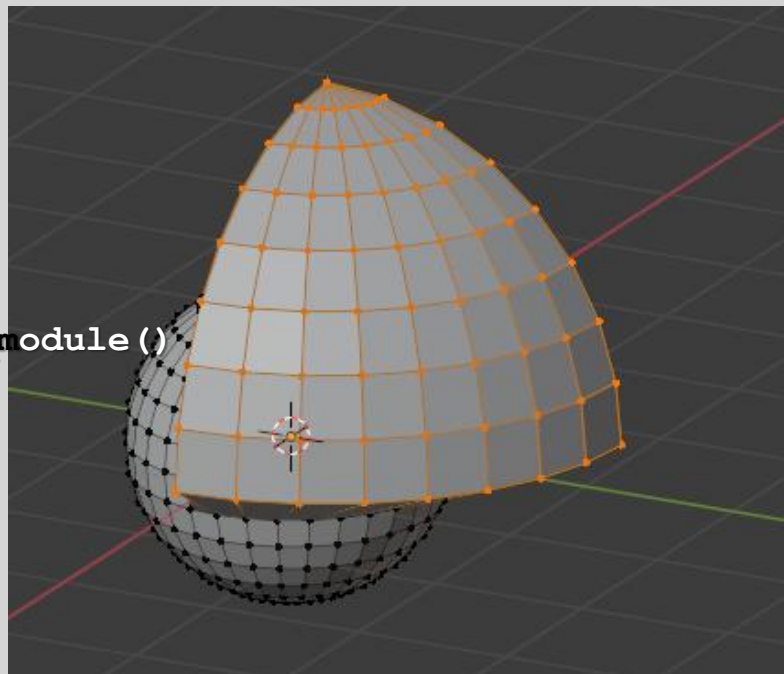


```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere('Sphere-1')
tk.sel.scale((2,)*3)
tk.mode("EDIT")

# Selects upper right quadrant of sphere
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'VERT', 'LOCAL')
tk.sel.translate((1, 1, 1))
tk.sel.scale((2, 2, 2))
```

(2,)\*3



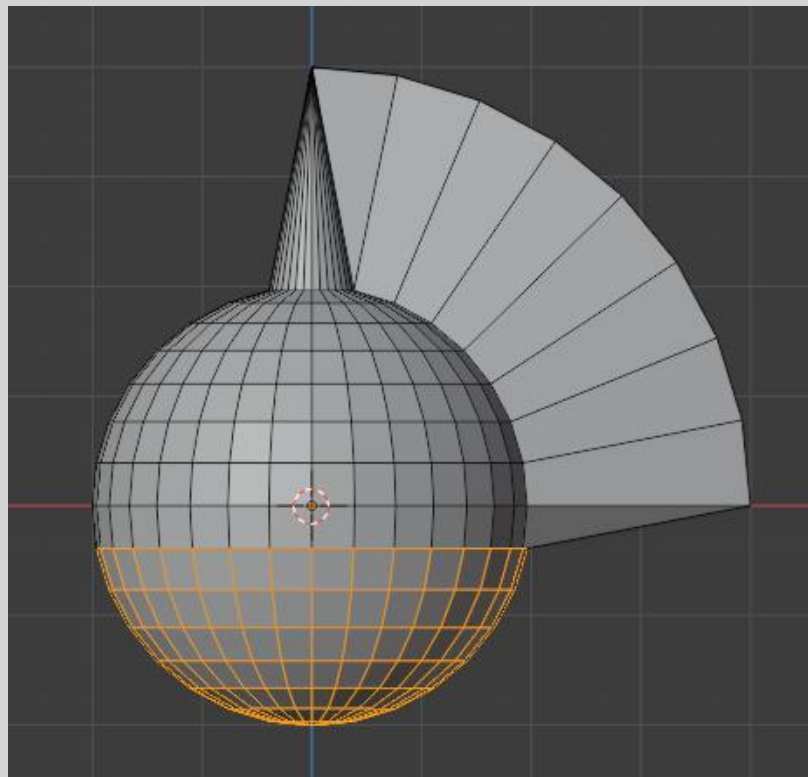


## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

# Selects a slice of lower half of sphere

```
tk.act.select_by_loc((-2, -2, -2), (2, 2, -0.3), 'EDGE', 'GLOBAL')
```







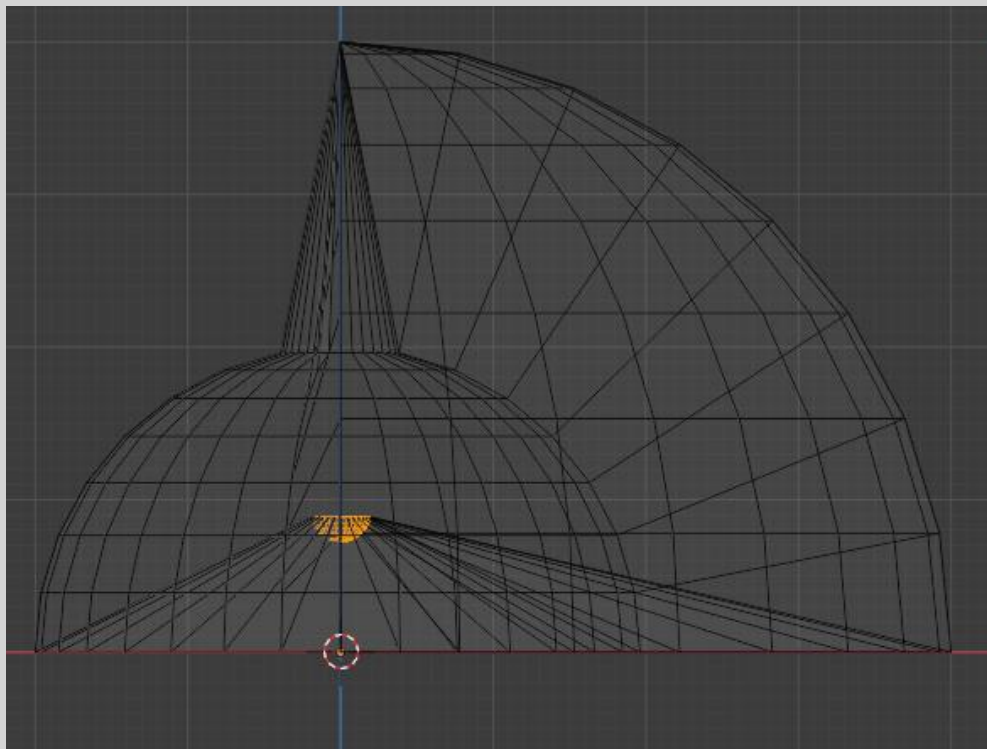
## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

```
# Mess with it
```

```
tk.sel.translate((0, 0, 2))
```

```
tk.sel.scale((0.1, 0.1, 0.1))
```

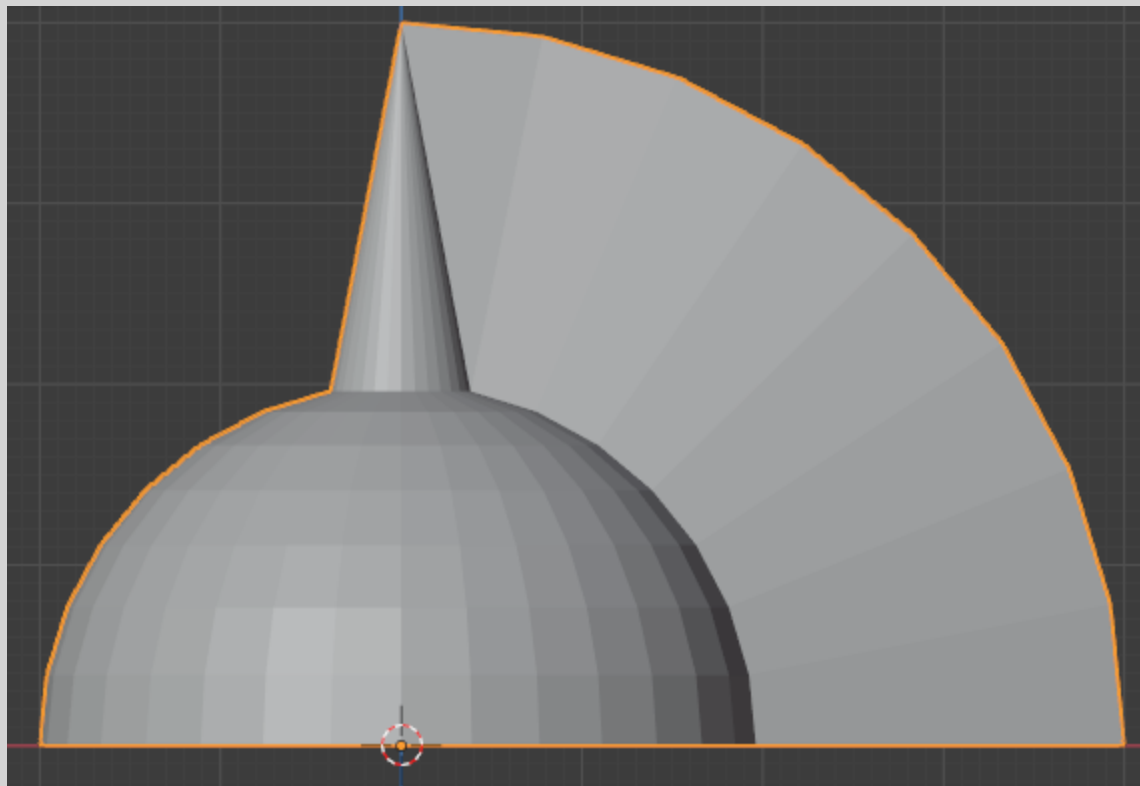




## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

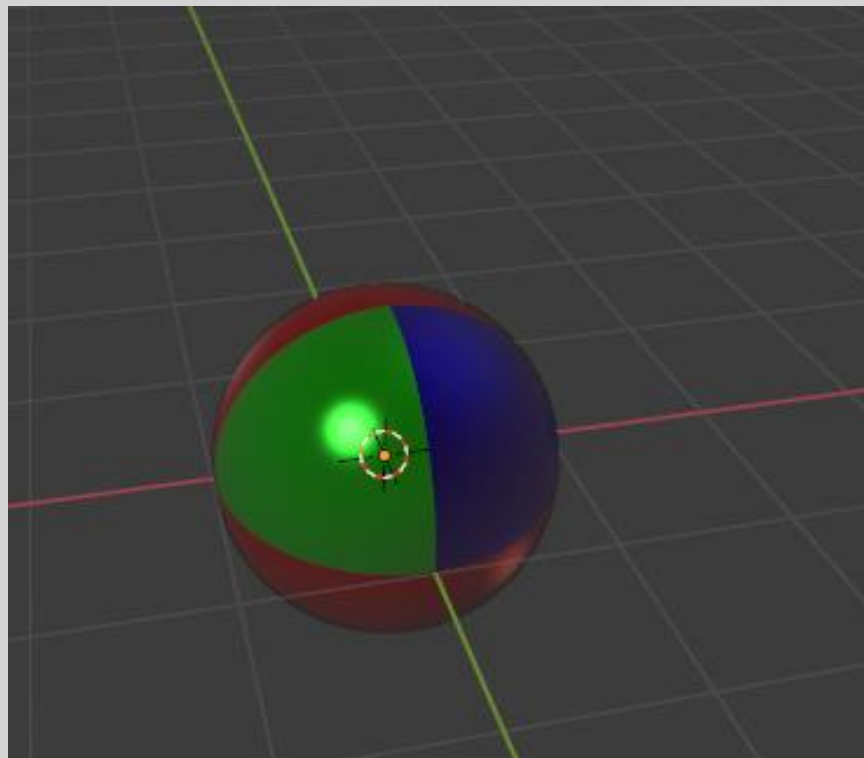
```
tk.mode('OBJECT')
```





# Blender/Python API

## Selecting and Coloring of a Sphere







## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

```
import bpy
import bmesh
from math import radians
tk = bpy.data.texts["bco602tk.py"].as_module()

# Create materials
alpha = 0.9
red = tk.makeMaterial('Red', (1, 0, 0, alpha), 0.5, 0, 0.1)
green = tk.makeMaterial('Green', (0, 1, 0, alpha), 0.5, 0, 0.1)
blue = tk.makeMaterial('Blue', (0, 0, 1, alpha), 0.8, 0.8, 0.2)

# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()
tk.create_sphere("sphere_1")
```



## Blender/Python API

### Selecting and Transforming Pieces of a Sphere

```
ob = bpy.context.object  
me = ob.data  
me.materials.append(red)  
me.materials.append(blue)  
me.materials.append(green)
```

```
tk.mode("EDIT")  
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'FACE', 'GLOBAL')  
# use second material slot  
ob.active_material_index = 1  
bpy.ops.object.material_slot_assign()
```

Same selection

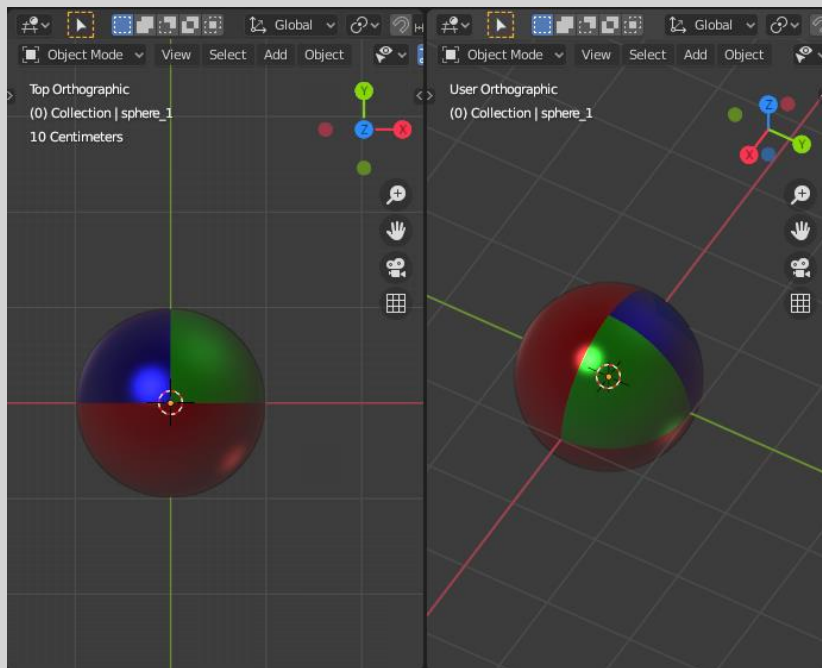
```
tk.mode("OBJECT")  
tk.sel.rotate_z(radians(90))  
tk.mode("EDIT")  
tk.act.select_by_loc((0, 0, 0), (1, 1, 1), 'FACE', 'GLOBAL')  
ob.active_material_index = 2  
bpy.ops.object.material_slot_assign()
```

90 degree

# Blender/Python API

## Selecting and Transforming Pieces of a Sphere

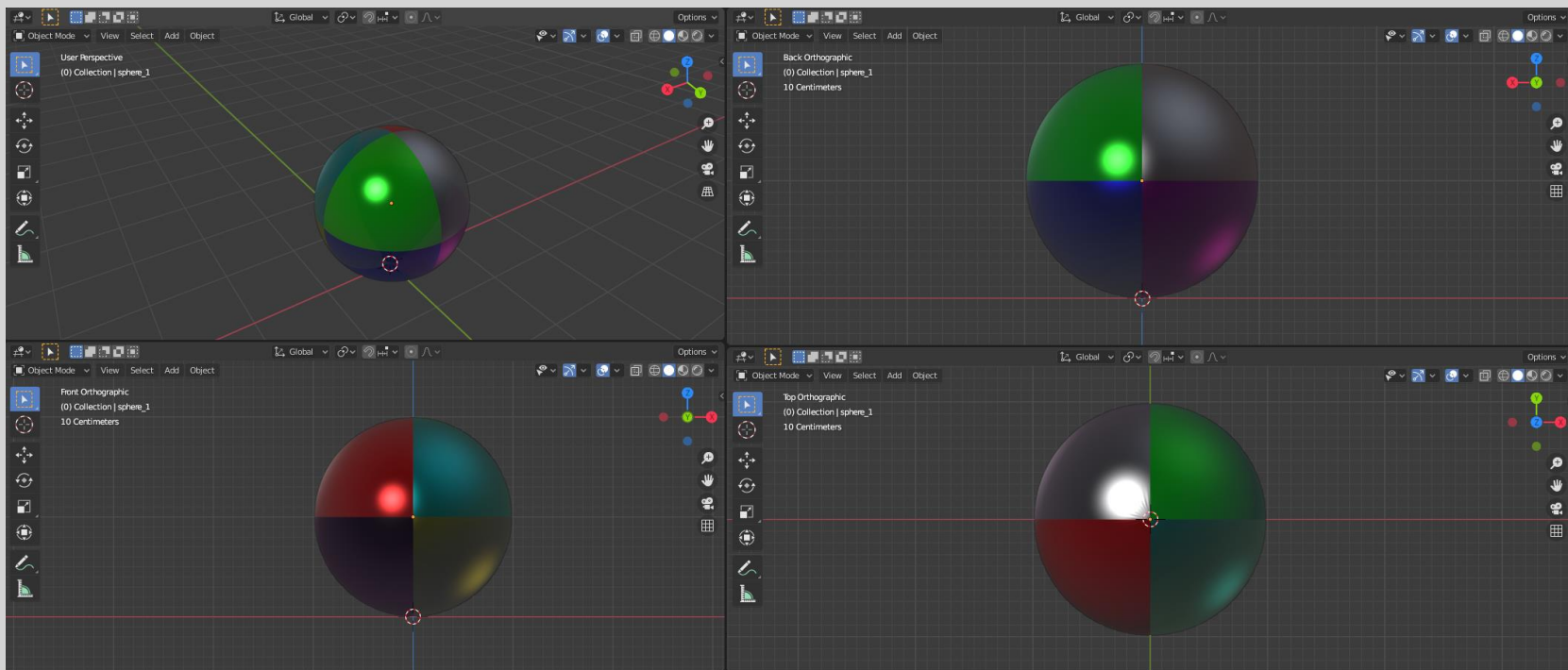
```
tk.mode("OBJECT")
```



# Blender/Python API

## Selecting and Transforming Pieces of a Sphere

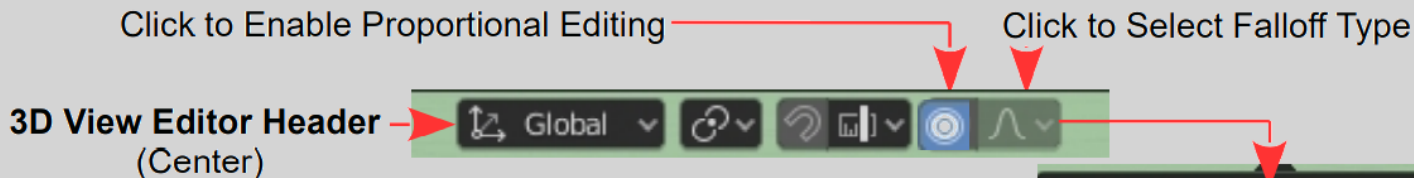
Classwork : Color all 8 quadrant of the sphere  
10 minutes break





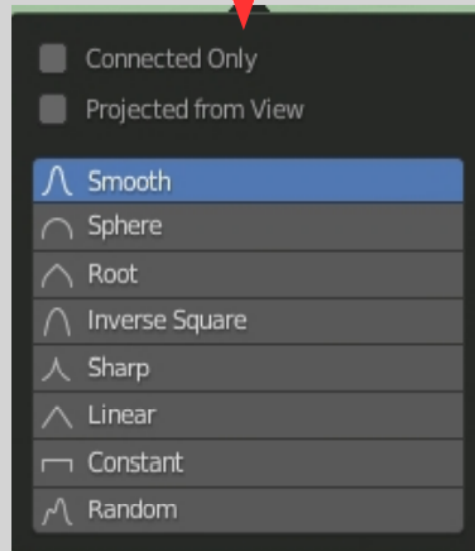
## Blender/Python API Proportional Vertex Editing

**Proportional Vertex Editing** is used to create a flow in the shape when editing Vertices. To turn **Proportional Vertex Editing** on, in **Edit mode**, click the **Proportional Editing** button in the **3D View Editor Header**



**Note:** You can only select Proportional Editing while in Edit Mode.

With Proportional Vertex Editing enabled there are several types of Falloff available. To see what this means perform the following:

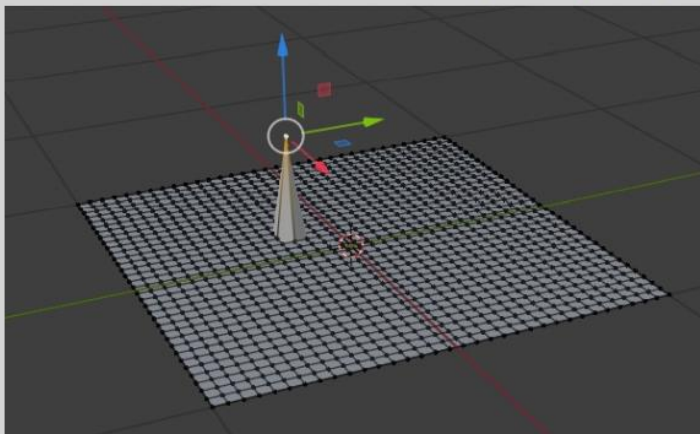




## Blender/Python API Proportional Vertex Editing

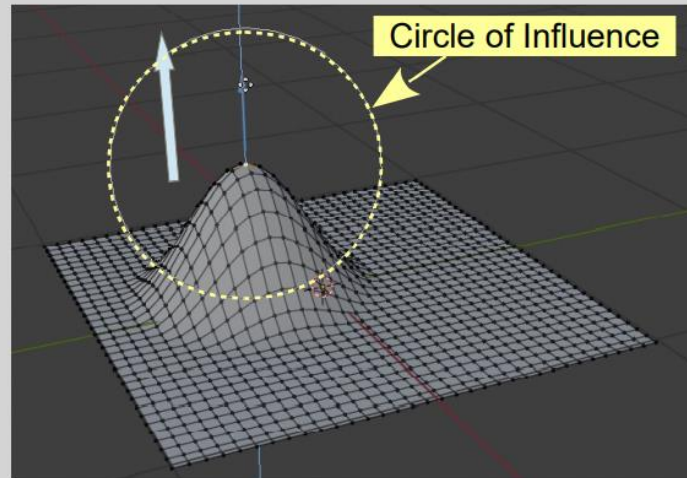
**Note:** To adjust the **Circle of Influence** select a Vertex or Vertices. Press the **G** Key (grab) and scroll the Mouse Wheel.  
(See over page)

Select a single Vertex and Translate up on the Z Axis



Plane Subdivided – Single Vertex Translated

**Note:** Proportional Vertex Editing has **NOT** been activated.



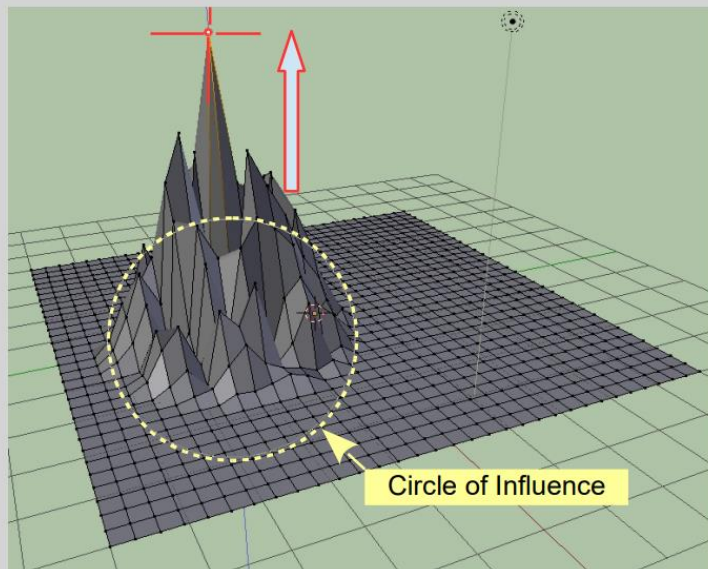
Single Vertex Translated on the Z Axis  
**Proportional Vertex Editing Activated**





## Blender/Python API Proportional Vertex Editing

The diameter of the circle is adjustable. Select the Vertex to be Translated. Press the G Key (places the Vertex in Grab Mode) to display the circle. Without moving the Mouse, scroll the mouse wheel. Click LMB. The circle display is cancelled but the influence is retained. Pressing the G Key and moving the Mouse moves the selected Vertex. Pressing the Grab button in the Tool Panel displays the Manipulation Widget and allows Translation of the Vertex. Although not displayed the Circle of Influence previously set is in effect. Experiment with the different types of Falloff.



Single Vertex Translated with Random Falloff Selected



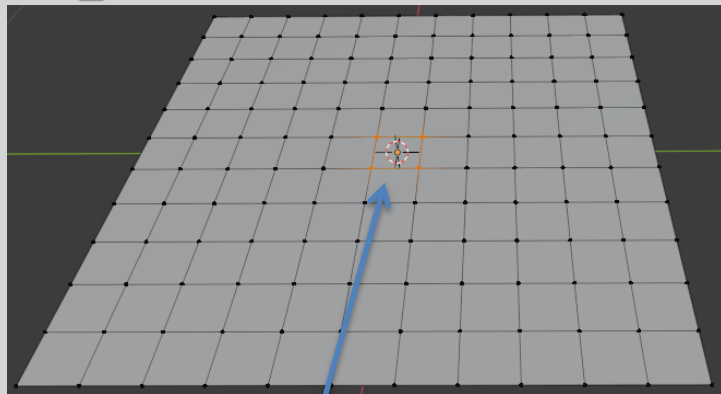
## Blender/Python API Proportional Vertex Editing

```
import bpy
import bmesh
tk = bpy.data.texts["bco602tk.py"].as_module()

fallOff = ['SPHERE', 'SMOOTH', 'ROOT', 'INVERSE_SQUARE', 'SHARP',
'LINEAR', 'CONSTANT', 'RANDOM']
# Will fail if scene is empty
tk.mode("OBJECT")
tk.delete_all()

for i, foff in enumerate(fallOff):

    tk.create.plane('Plane-{}'.format(i))
    tk.mode("EDIT")
    tk.act.select_by_loc((-1, -1, 0), (1, 1, 0), 'EDGE', 'LOCAL')
    bpy.ops.mesh.subdivide(number_cuts=10)
    bpy.ops.mesh.select_all(action = "DESELECT")
    tk.act.select_by_loc((-0.1, -0.1, 0), (0.1, 0.1, 0), 'VERT', 'LOCAL')
```



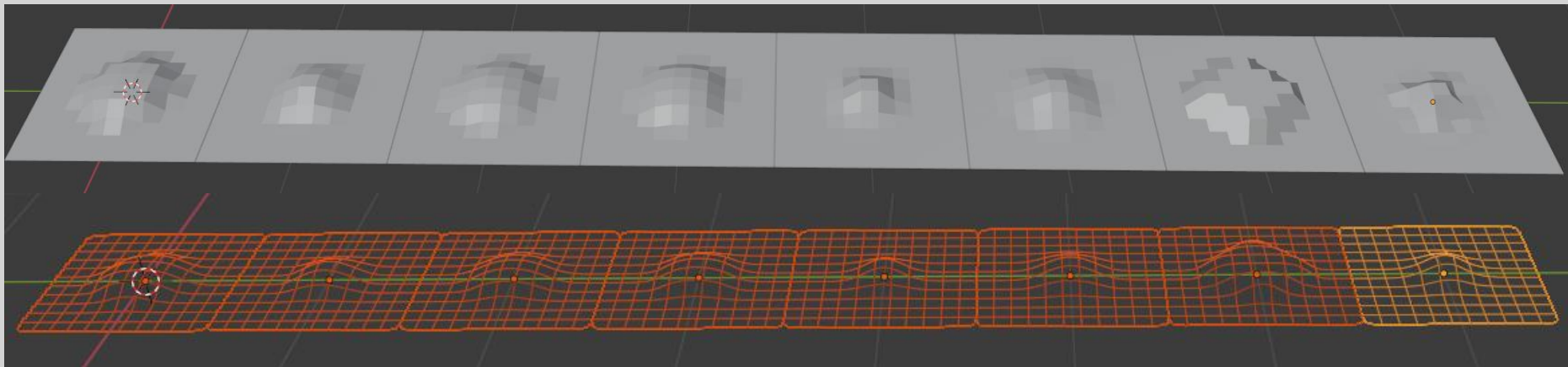
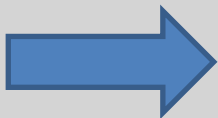




## Blender/Python API Proportional Vertex Editing

```
bpy.ops.transform.translate(value=(0, 0, 0.1), \
    use_proportional_edit = True, \
    proportional_edit_falloff = 'FOFF', \
    proportional_size = 0.2, \
    use_proportional_connected = False, \
    use_proportional_projected = False)

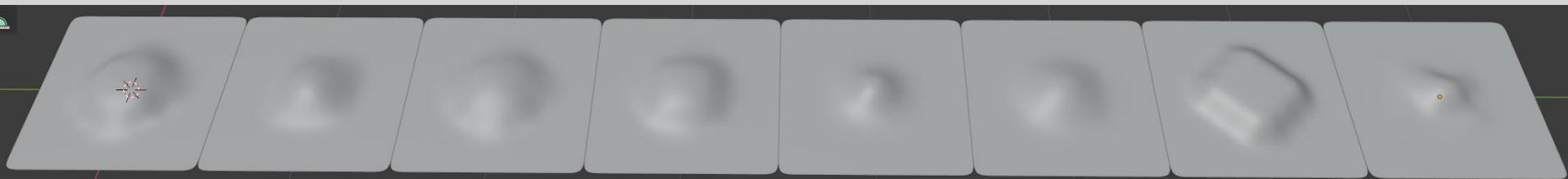
tk.mode('OBJECT')
tk.sel.translate((0, i, 0))
```





## Blender/Python API Proportional Vertex Editing

```
tk.setSmooth(bpy.context.object, level = 3, smooth = True)
```



'SPHERE', 'SMOOTH', 'ROOT', 'INVERSE\_SQUARE', 'SHARP', 'LINEAR', 'CONSTANT', 'RANDOM'

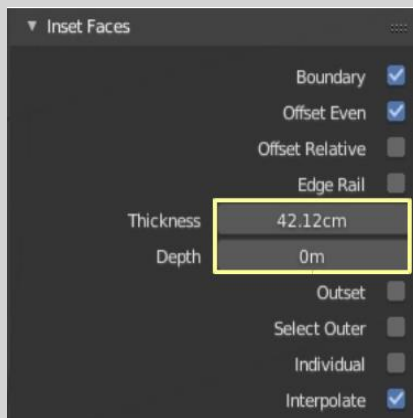
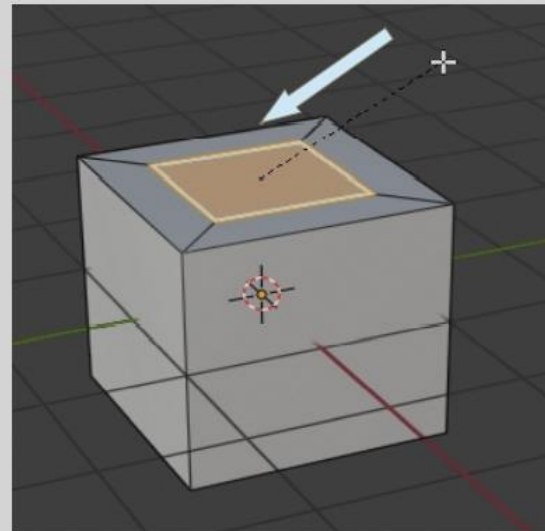


## Blender/Python API Inset Faces

**Inset Faces** creates new Faces inside a selected geometry.

To demonstrate, select the top **Face** of the default Cube while in Edit Mode. Press the **I Key** and move the **Mouse Cursor** towards the center of the selected Face. You see new Faces created. Click LMB to set the new Faces in position.

The **Thickness slider** controls the size of the new Face. The **Depth slider** displaces the Face normal to the surface (positive values above the surface, negative values below).

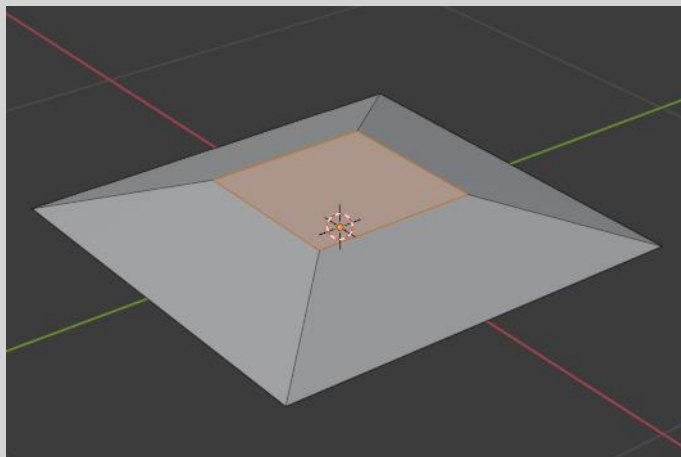




## Blender/Python API Inset Faces

```
import bpy
import bmesh
tk = bpy.data.texts["bco602tk.py"].as_module()

tk.mode("OBJECT")
tk.delete_all()
tk.create.plane('Plane')
tk.mode("EDIT")
tk.act.select_by_loc((-1, -1, 0), (1, 1, 0), 'FACE', 'LOCAL')
bpy.ops.mesh.inset(thickness=0.3, depth=0.1, release_confirm = True)
```





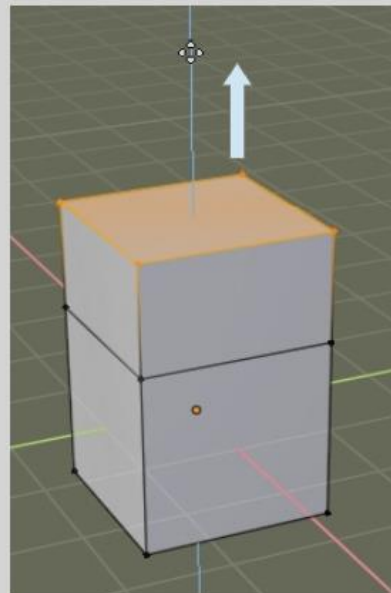
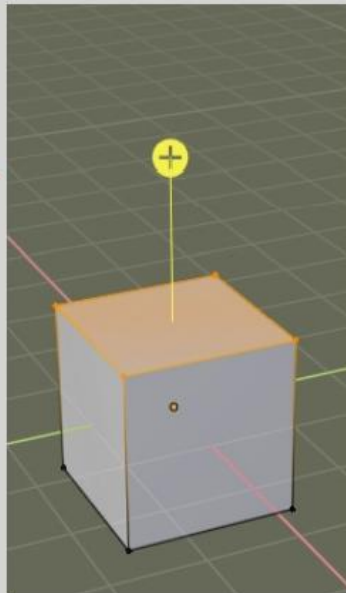
## Blender/Python API Extrude Region

### Extrude Region



**Extrude Along Normal**  
means Extrude at right angles to the selected Face.

**Extrude Individual**  
allows Extrusion of an individual Face.

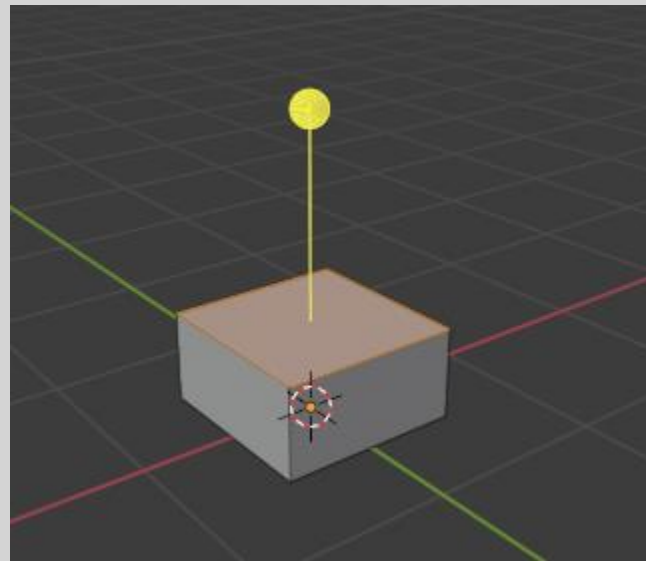




## Blender/Python API Extrude Region

```
import bpy
import bmesh
tk = bpy.data.texts["bco602tk.py"].as_module()

tk.mode("OBJECT")
tk.delete_all()
tk.create.plane('Plane')
tk.mode("EDIT")
tk.act.select_by_loc((-1, -1, 0), (1, 1, 0), 'FACE', 'LOCAL')
bpy.ops.mesh.extrude_region_move(TRANSFORM_OT_translate={"value": (0, 0, 0.5)})
```

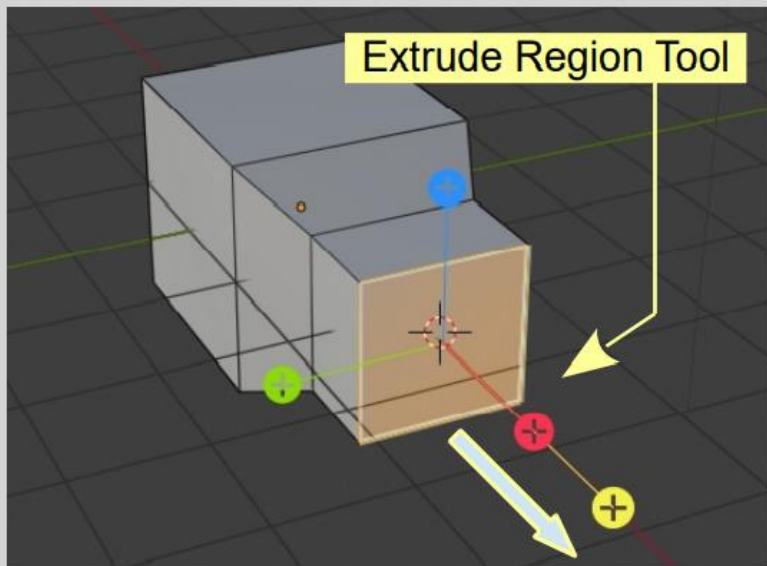




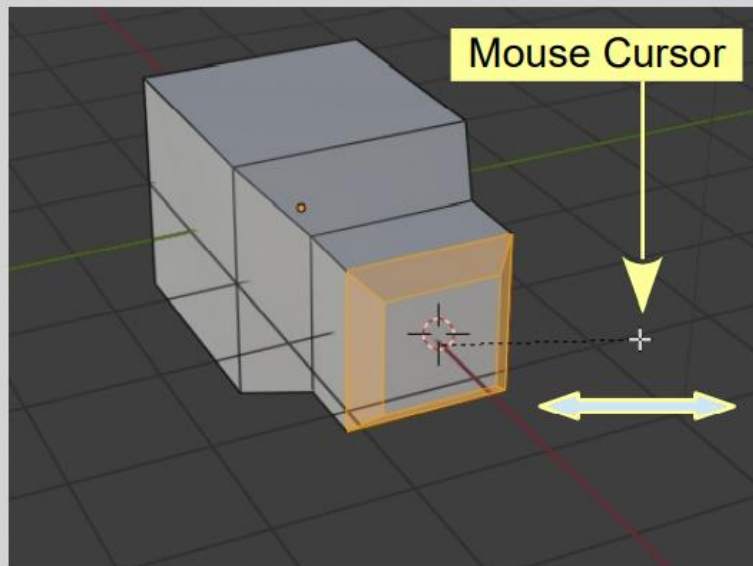


## Blender/Python API Bevel

The **Bevel Tool** bevels the Edges of a selected Face.



The default Cube with the front face Extruded and Scaled down then Extruded a second time.



With the face selected activate the Bevel Tool. Place the Mouse Cursor, click hold and drag to bevel the edges of the Face.

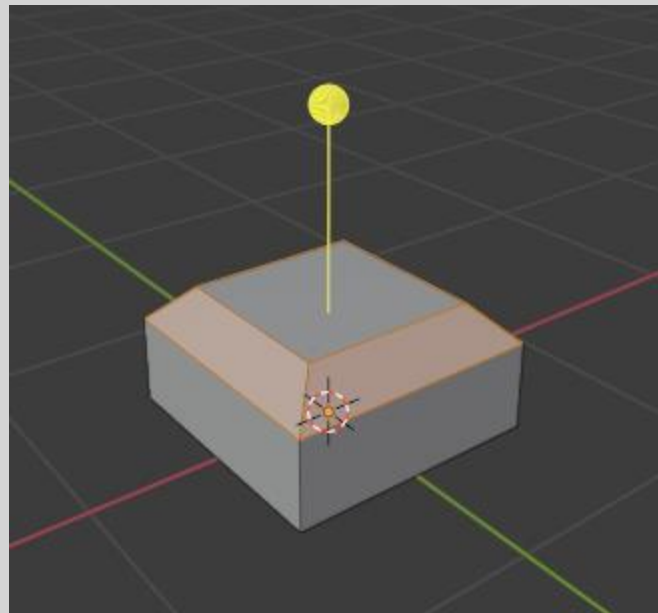


## Blender/Python API

### Bevel

```
import bpy
import bmesh
tk = bpy.data.texts["bco602tk.py"].as_module()

tk.mode("OBJECT")
tk.delete_all()
tk.create.plane('Plane')
tk.mode("EDIT")
tk.act.select_by_loc((-1, -1, 0), (1, 1, 0), 'FACE', 'LOCAL')
bpy.ops.mesh.extrude_region_move(TRANSFORM_OT_translate={"value": (0, 0, 0.5)})
bpy.ops.mesh.bevel(offset = 0.15)
```





# Blender/Python API

## Scene Lighting

The default Blender Scene contains a single **Point Light Object**.

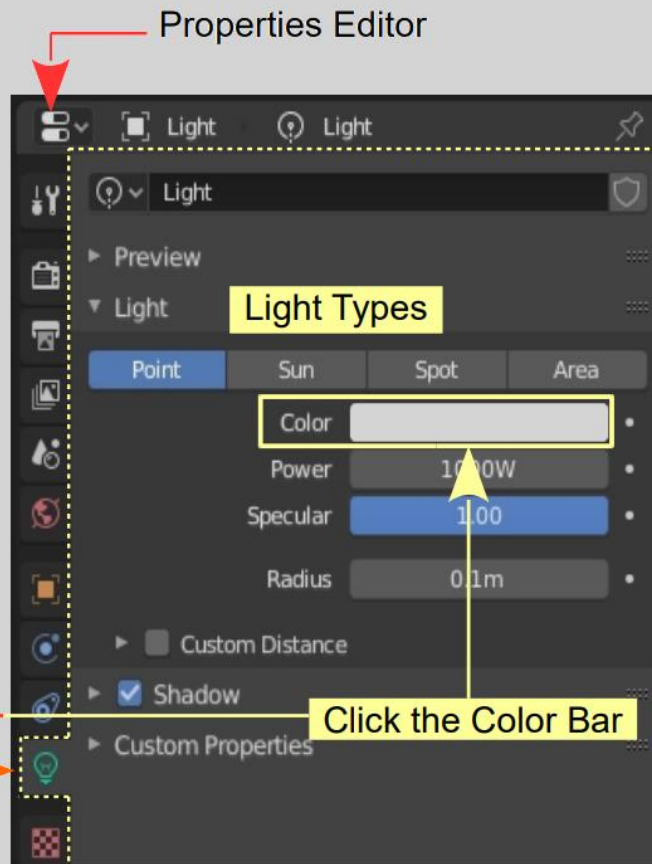
With the Light selected in the 3D Viewport Editor, go to the **Properties Editor, Object Data buttons** to display the setting options. You may change the Light by selecting one of the types in the **Light Tab**.

The Properties Editor display will vary depending on the Light Type selected.

The color of light may be selected by clicking the color bar to display a color picker circle.



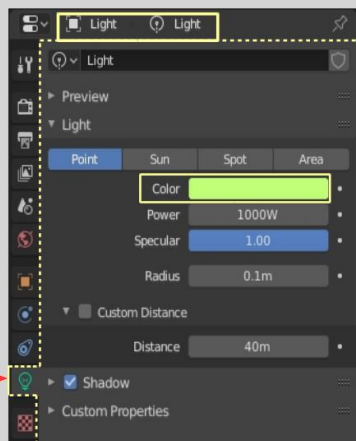
Object Data Button →



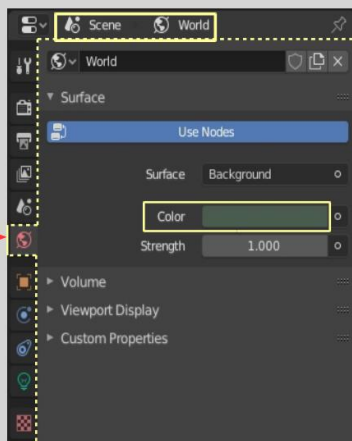
## Blender/Python API

### Scene Lighting

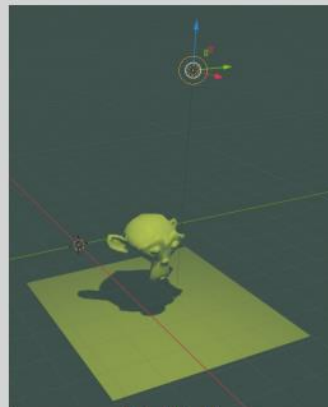
Properties Editor



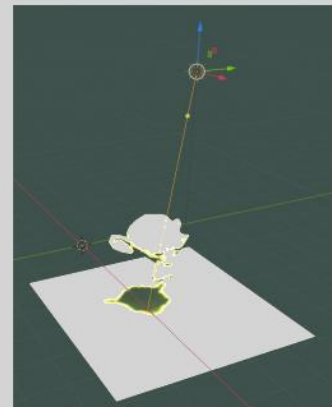
Object Data button – Light Color



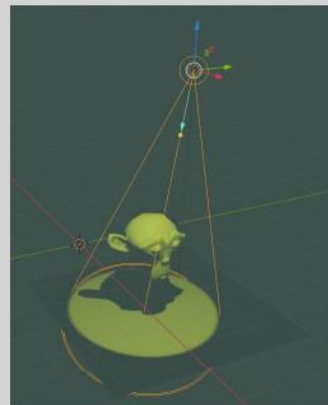
World button – Scene Background Color



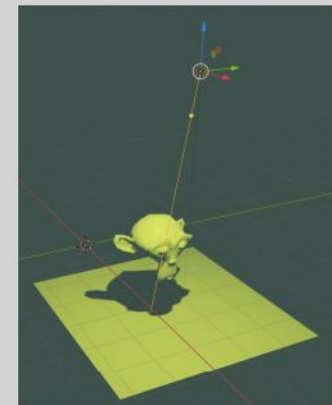
Point Light



Sun Light



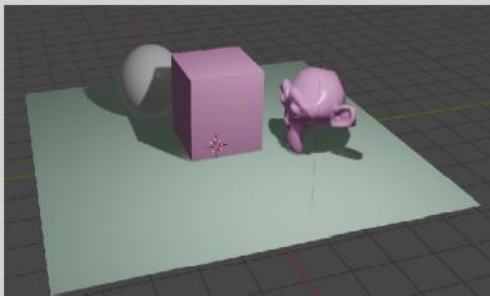
Spot Light



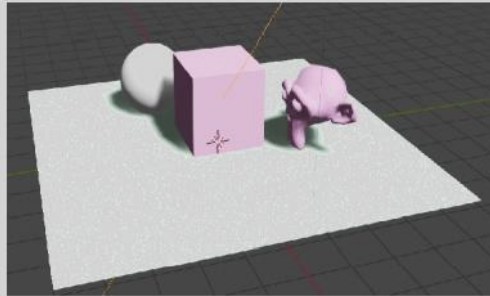
Area Light



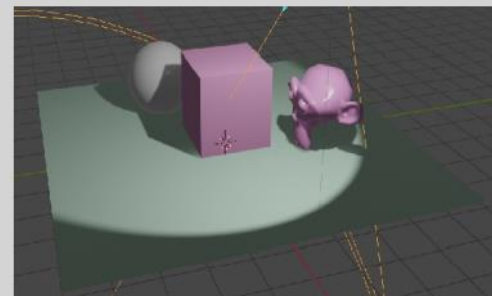
## Blender/Python API Scene Lighting



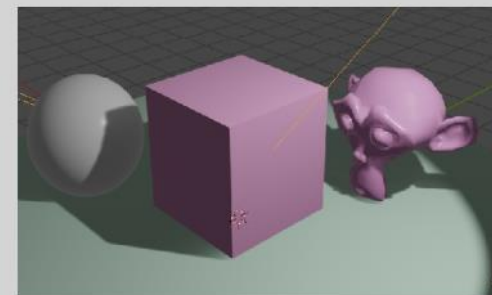
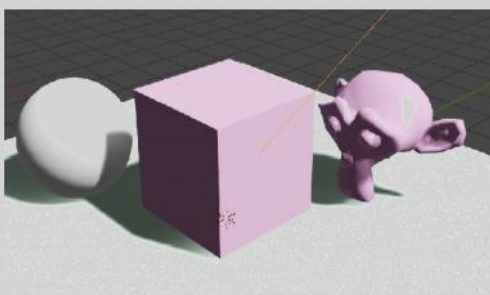
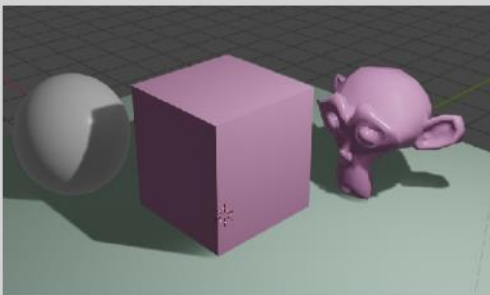
Point Lamp



Sun Lamp



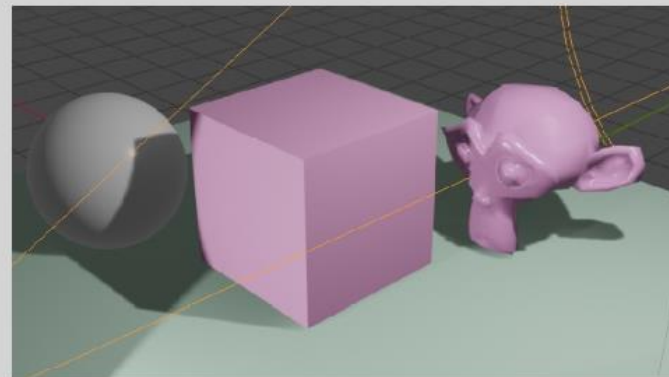
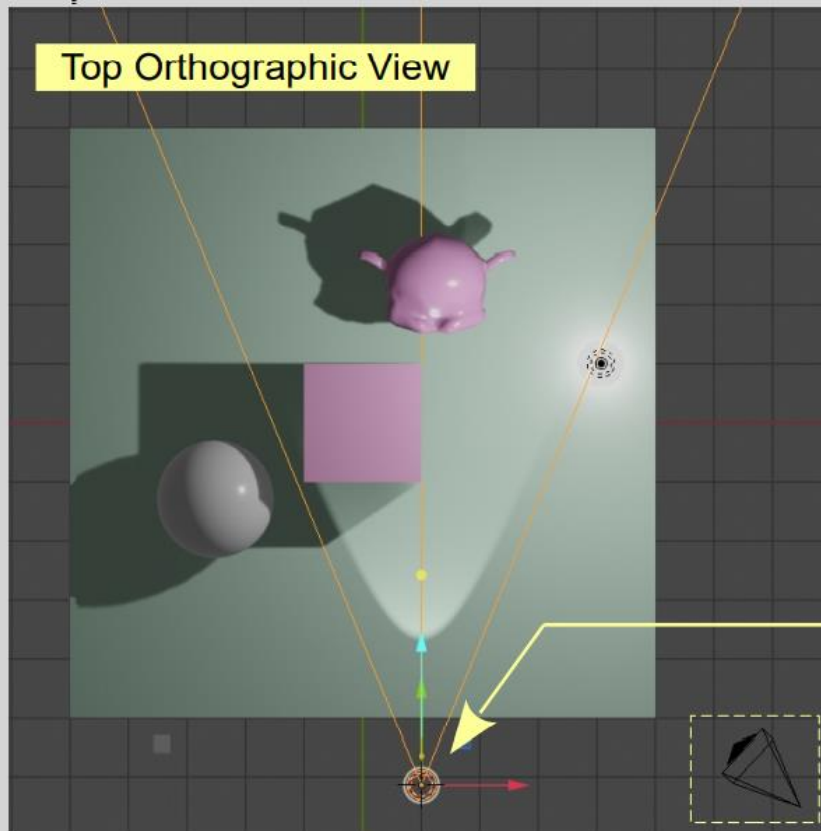
Spot Lamp



The upper row of images show part of the 3D Viewport Editor. The lower row shows Camera View. By leaving the default Point Light in position, then adding an additional Spot Light and directing it towards the left hand face of the Cube you remove the shadow on the Cube and the Monkey.



## Blender/Python API Scene Lighting



Additional Spot Light pointing towards the Cube and the Monkey.

The Camera in its default position.



## Blender/Python API Scene Lighting

```
def light(origin, type='POINT', energy = 1, color = (1,1,1)):  
    # Lamp types: 'POINT', 'SUN', 'SPOT', 'AREA'  
    #print('createLamp called')  
    bpy.ops.object.light_add(type = 'SPOT', radius = 1, location = origin)  
    obj = bpy.context.object  
    obj.data.type = type  
    obj.data.energy = energy  
    obj.data.color = color  
  
    return obj
```





## Blender/Python API Scene Lighting

```
import bpy
from math import radians
tk = bpy.data.texts["bco602tk.py"].as_module()

white = tk.makeMaterial('White', (1, 1, 1, 1), 0, 0, 0)

def light(origin, type='POINT', energy=1, color=(1,1,1)):
    .....
    return obj

tk.mode("OBJECT")
tk.delete_all()
tk.create.plane('Plane')
ob = bpy.context.object
me = ob.data
me.materials.append(white) # assign the plane
tk.act.scale((10,)*3)
```





## Blender/Python API Scene Lighting

```
redLamp = light((4, 0, 6), 'SPOT', 10000, (1, 0, 0))  
tk.act.rotation((radians(-0.242) , radians(33.6) , radians(-0.01)))
```

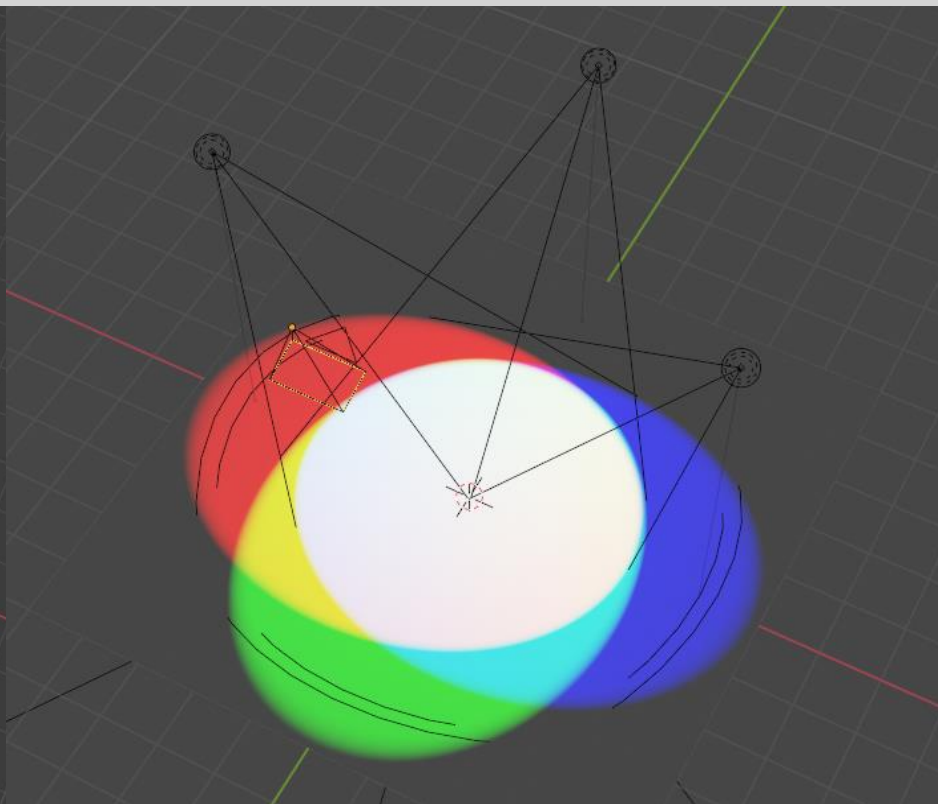
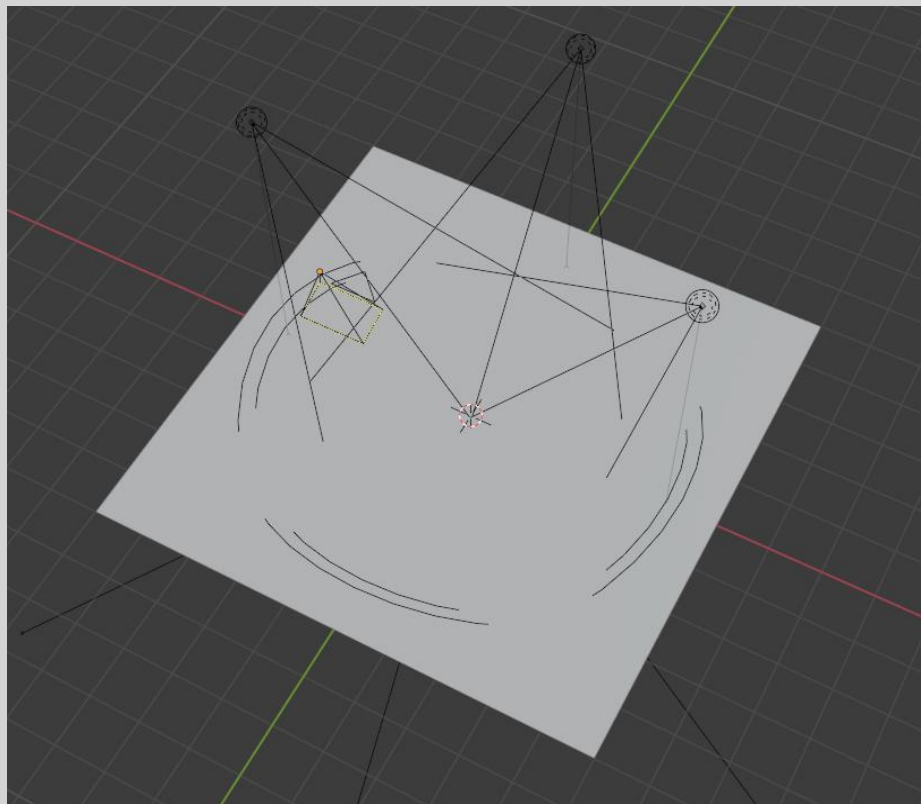
```
blueLamp = light((-4, 0, 6), 'SPOT', 10000, (0, 0, 1))  
tk.act.rotation((radians(-0.242) , radians(-33.6) , radians(0.01)))
```

```
greenLamp = light((0, 4, 6), 'SPOT', 10000, (0, 1, 0))  
tk.act.rotation((radians(-33.6) , radians(-0.242) , radians(0.01)))
```

```
bpy.ops.object.camera_add(location=(0, -4, 9) , rotation=((radians(15) ,  
radians(0) , radians(0))))
```

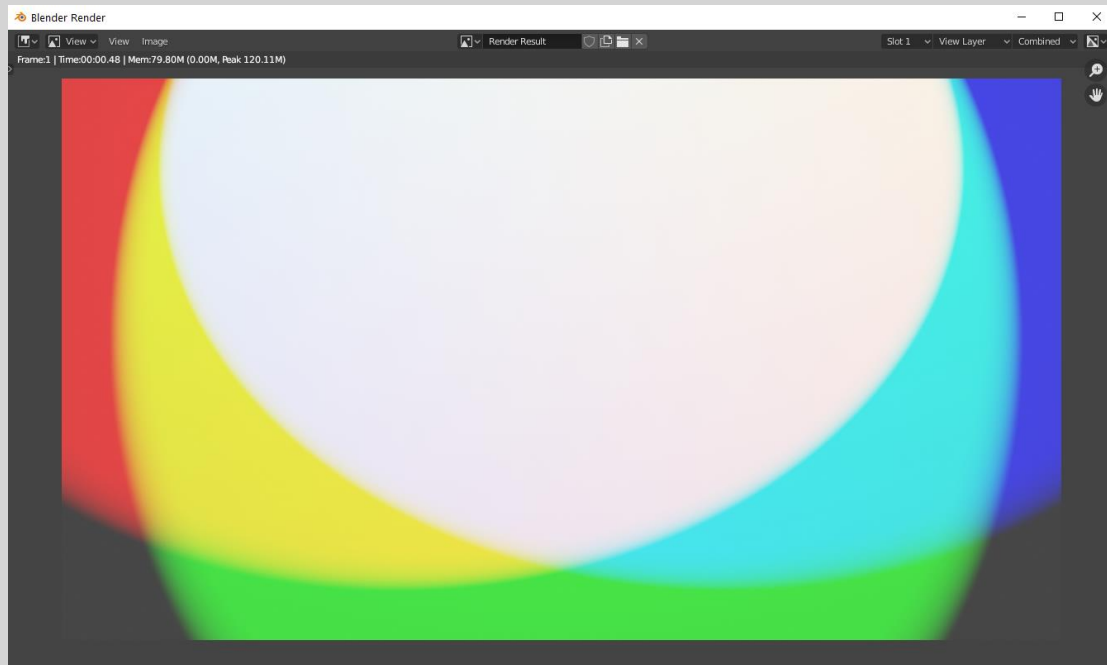
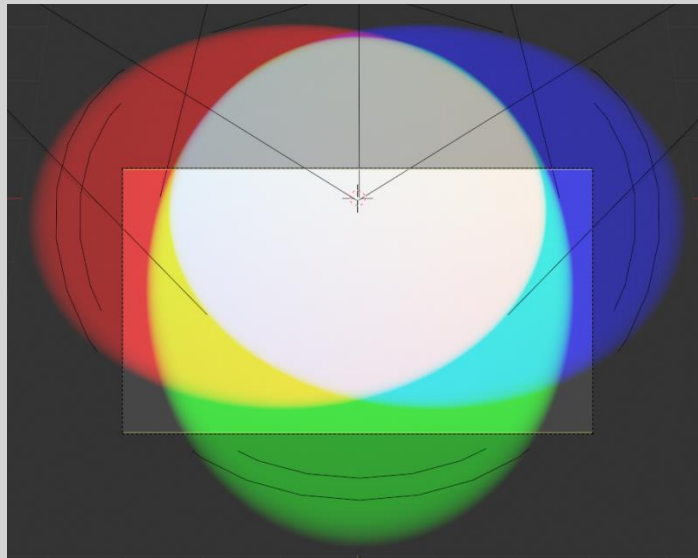
# Blender/Python API

## Scene Lighting



# Blender/Python API

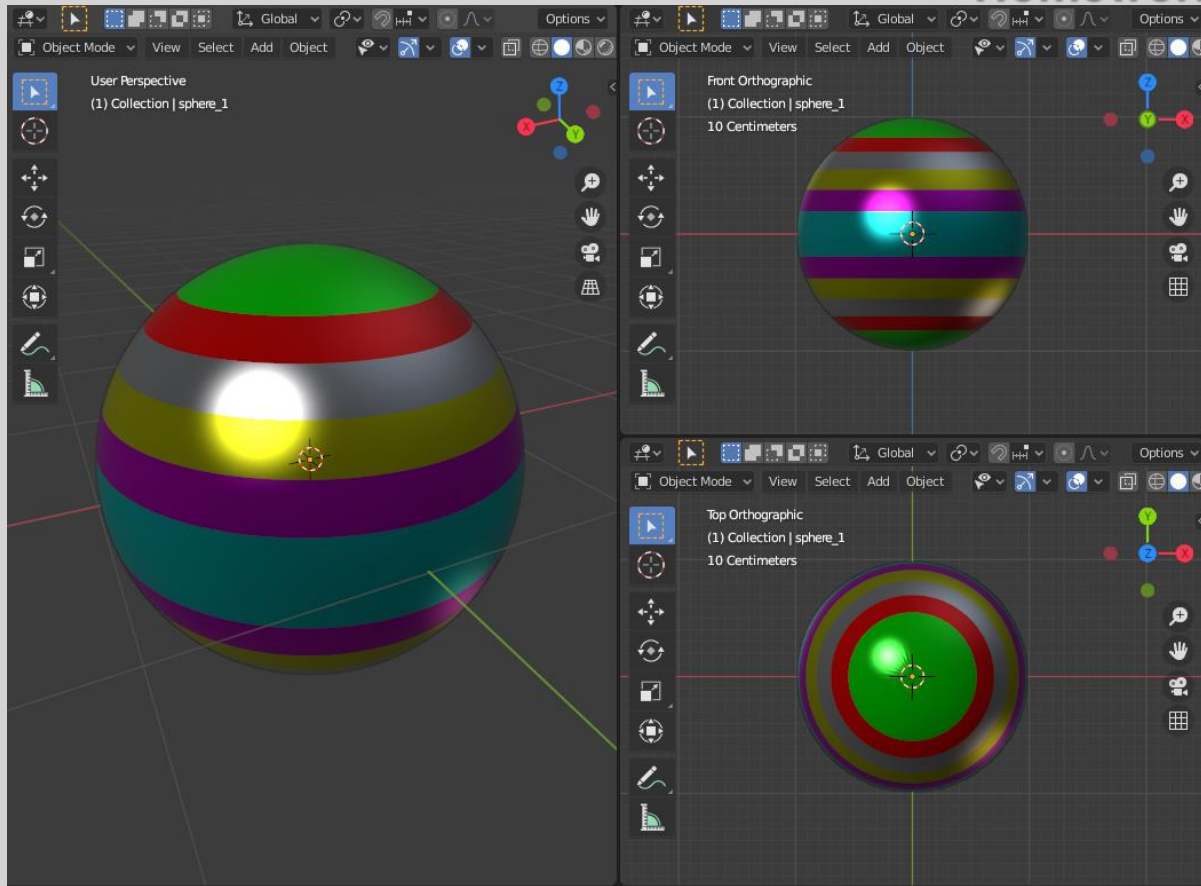
## Scene Lighting





## Blender/Python API Homework

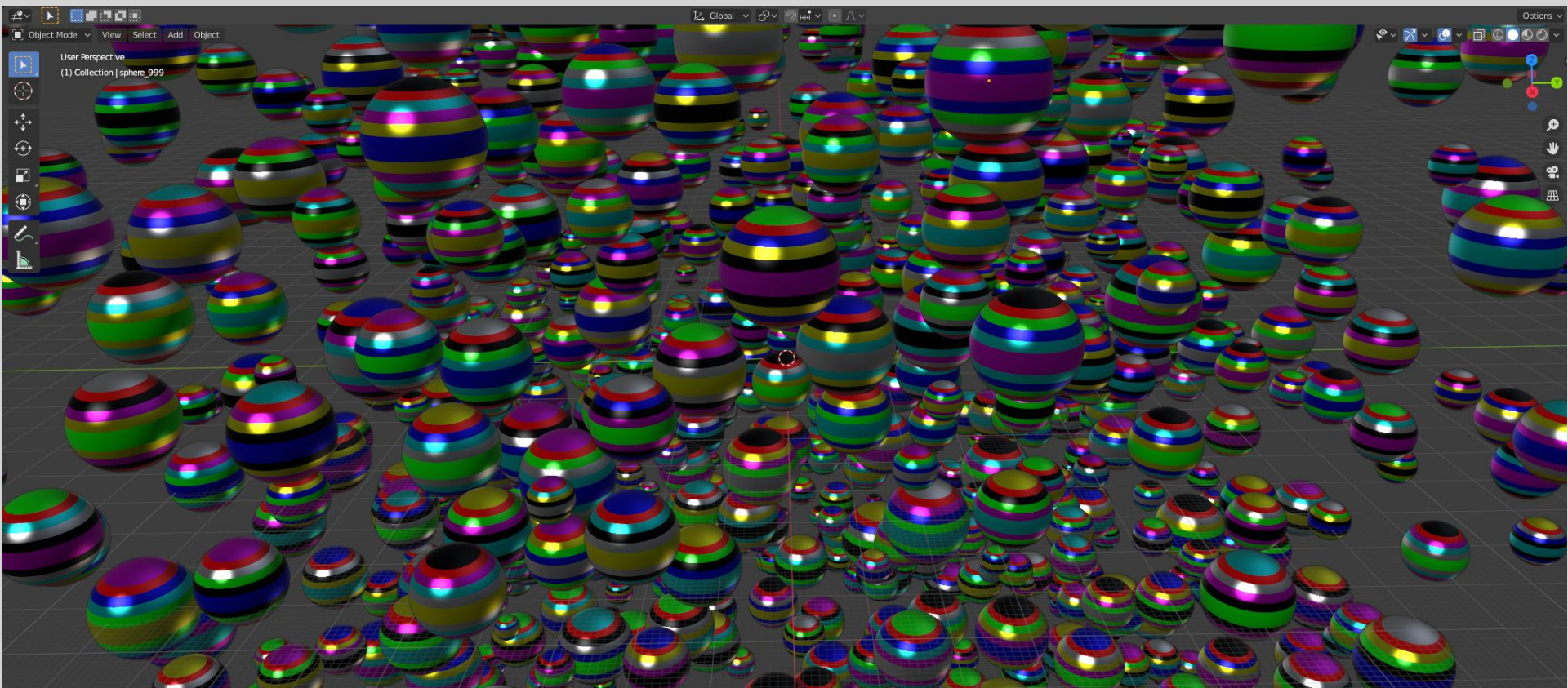
Create a sphere with  
random color stripes





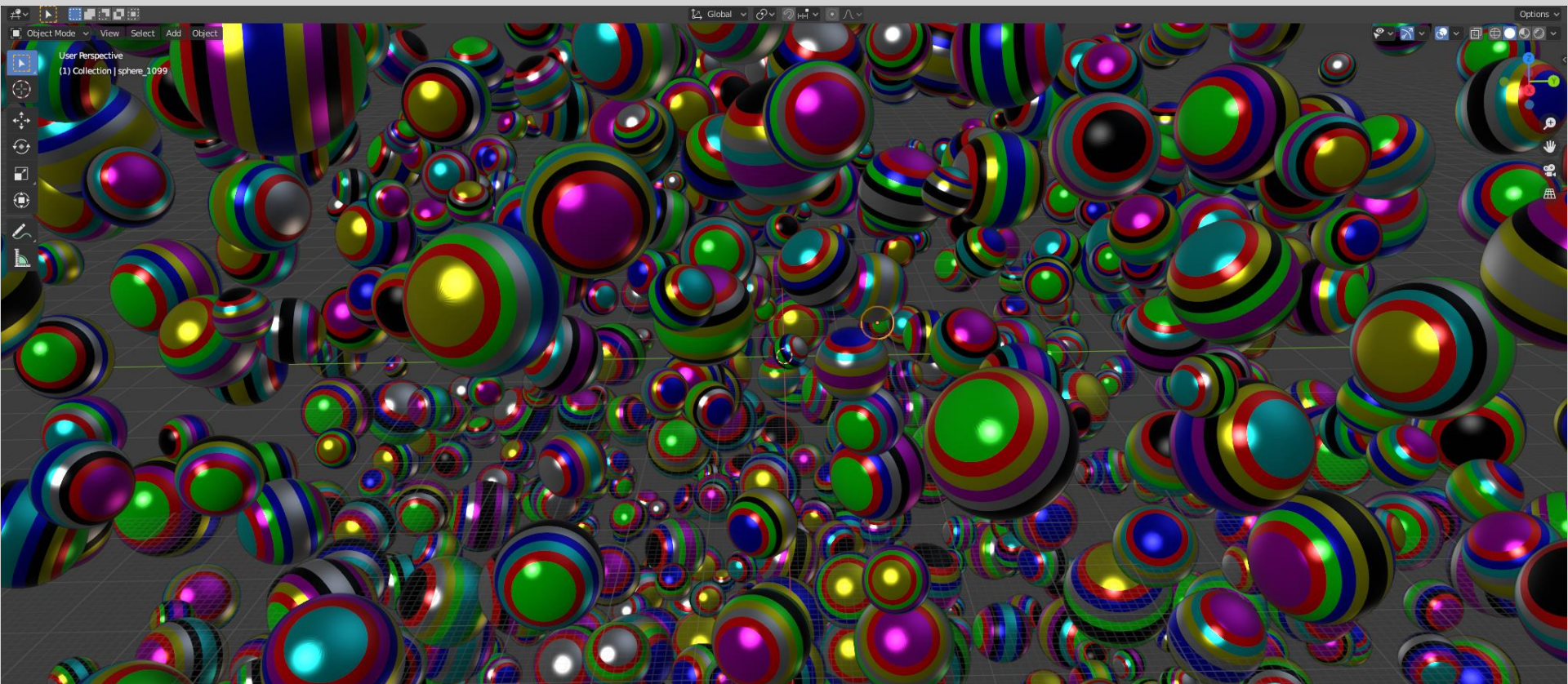
# Blender/Python API

## Homework





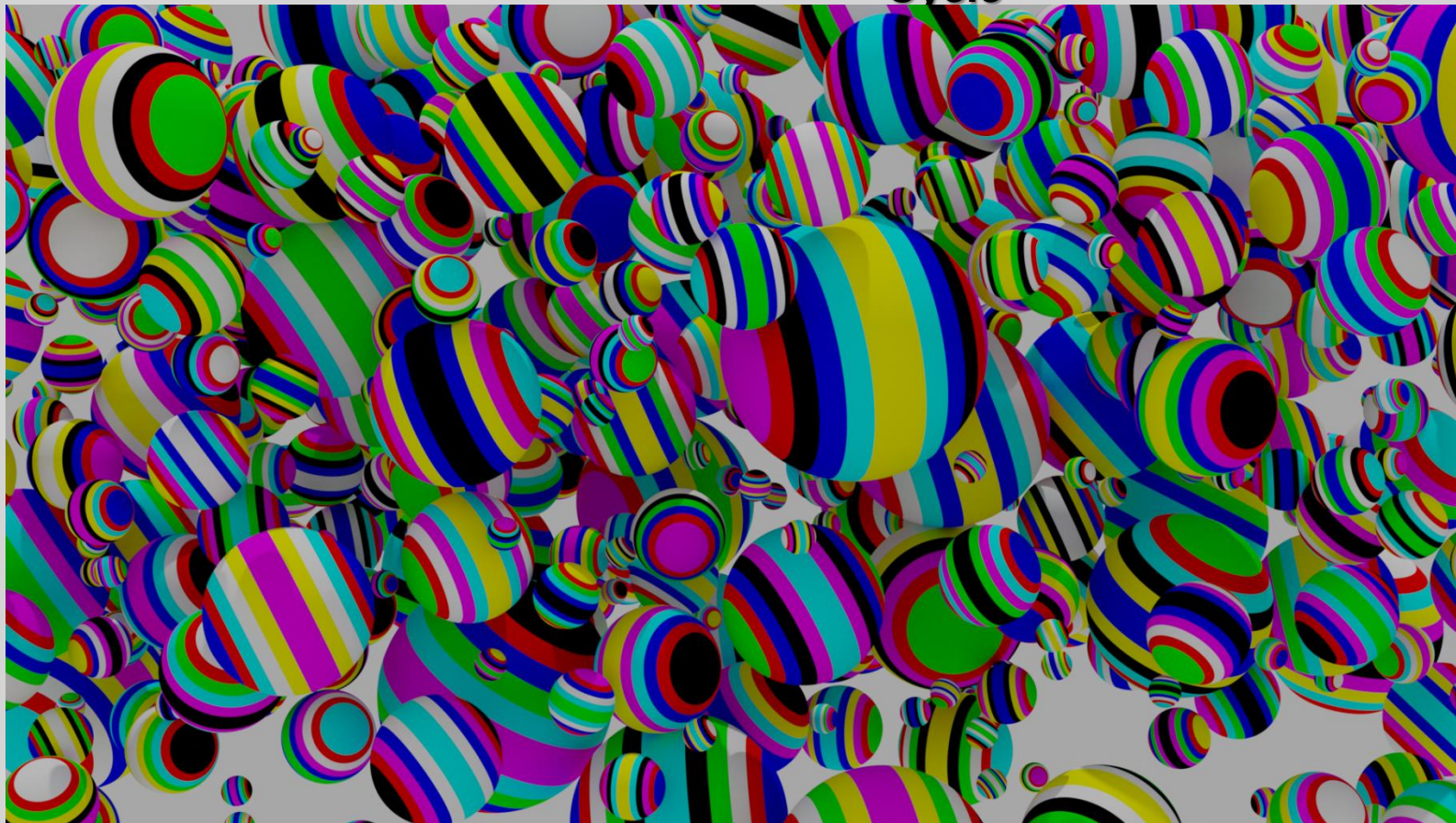
# Blender/Python API Homework







## Blender/Python API Cycle







# Blender/Python API

## Eevee

