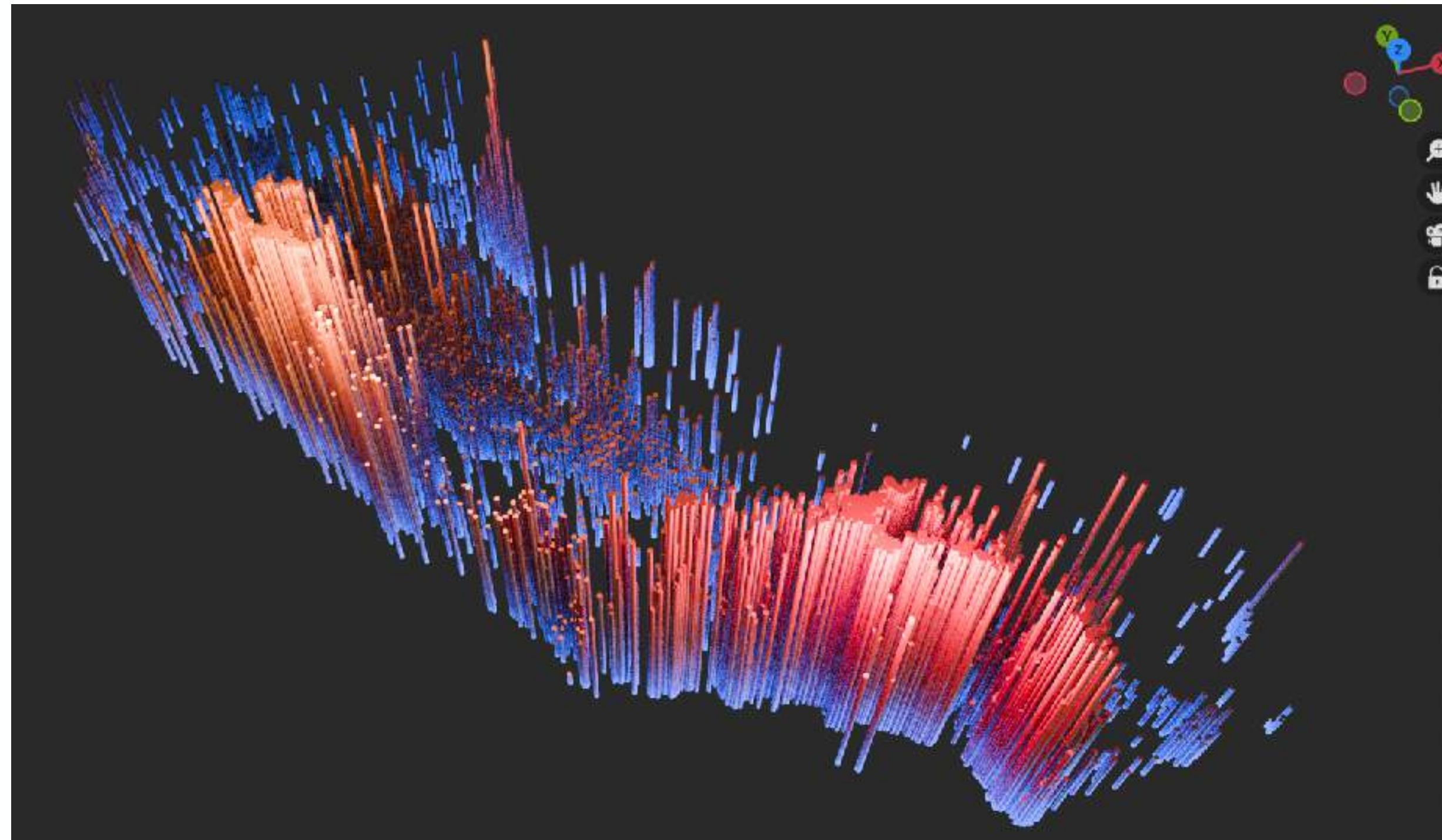




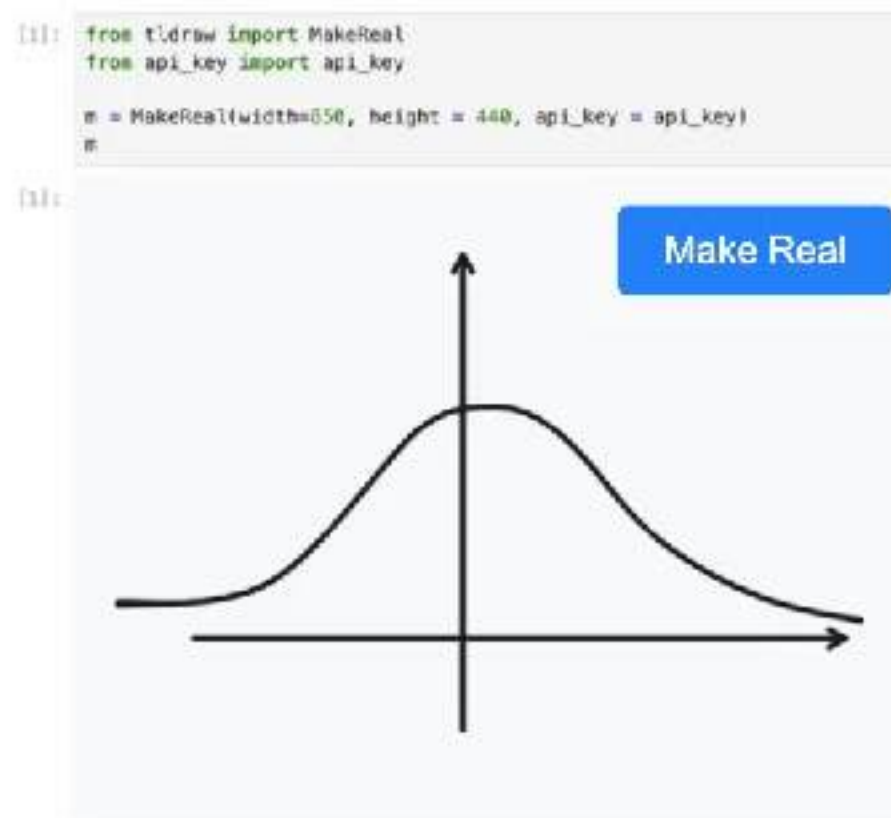
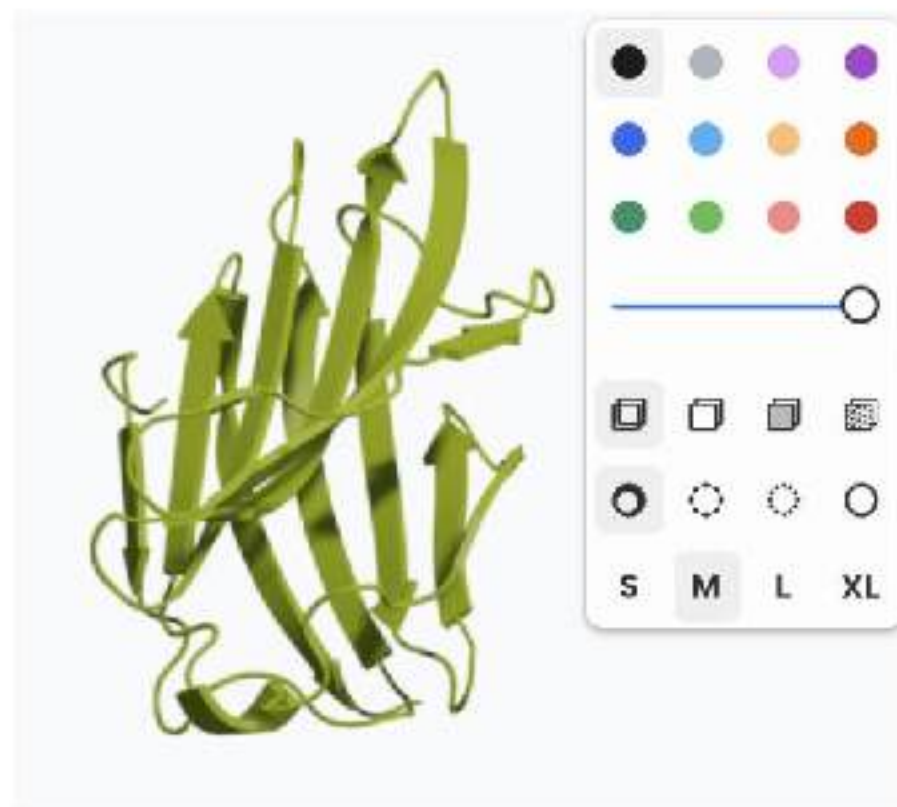
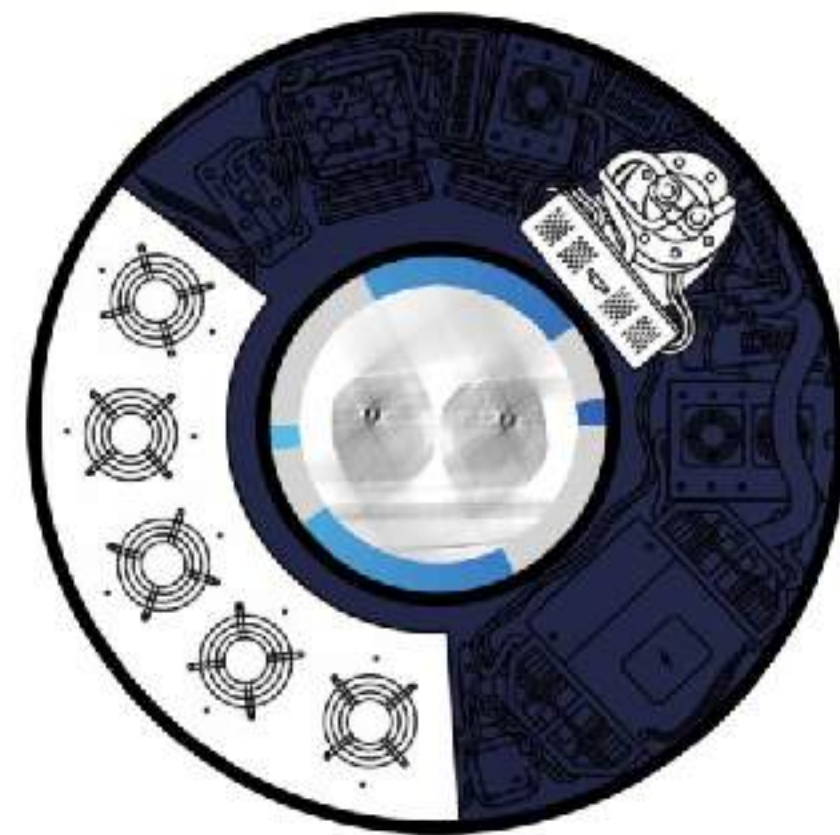
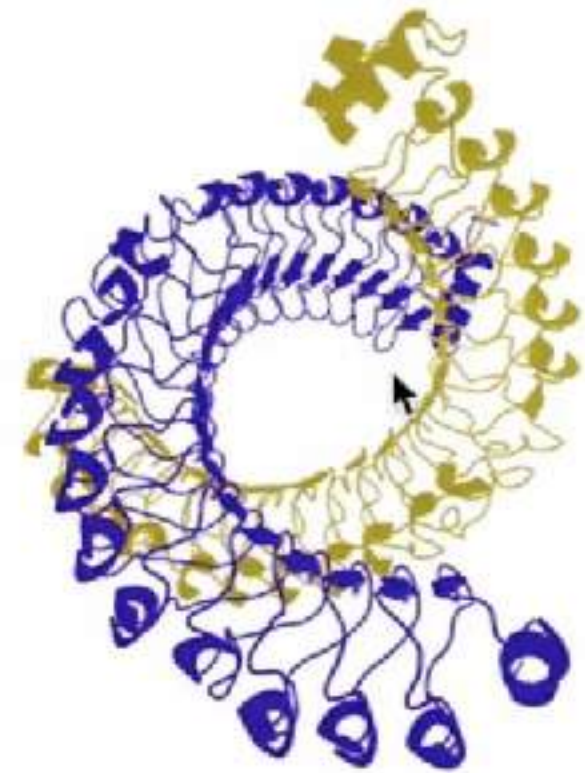
**Jan-Hendrik Müller**  
**16.01.2025,**  
**14:00-14:45**

## **Data visualisation using Blender within Jupyter notebooks**



# About me

- Biophysicist from Göttingen
- Open Source Contributor to Data visualization projects



**Jan-Hendrik Müller**  
**16.01.2025,**  
**14:00-14:45**

<https://www.linkedin.com/in/jan-hendrik-müller-765014209/>  
<https://bsky.app/profile/kolibril13.bsky.social>

**supported in 2024 by**



Federal Ministry  
of Education  
and Research

```
localhost
lab - JupyterLab
Circular Tree - NetworkX 3.3 documentation

File Edit View Run Kernel Tabs Settings Help

multiple_networks.ipynb
Code Notebook blender

[2]: import networkx as nx
import bpy

G = nx.balanced_tree(4, 4)
node_positions = nx.spring_layout(G, dim=3, scale=1.9)
edges = list(G.edges)
draw_network(node_positions, edges, sphere_radius = 0.05)

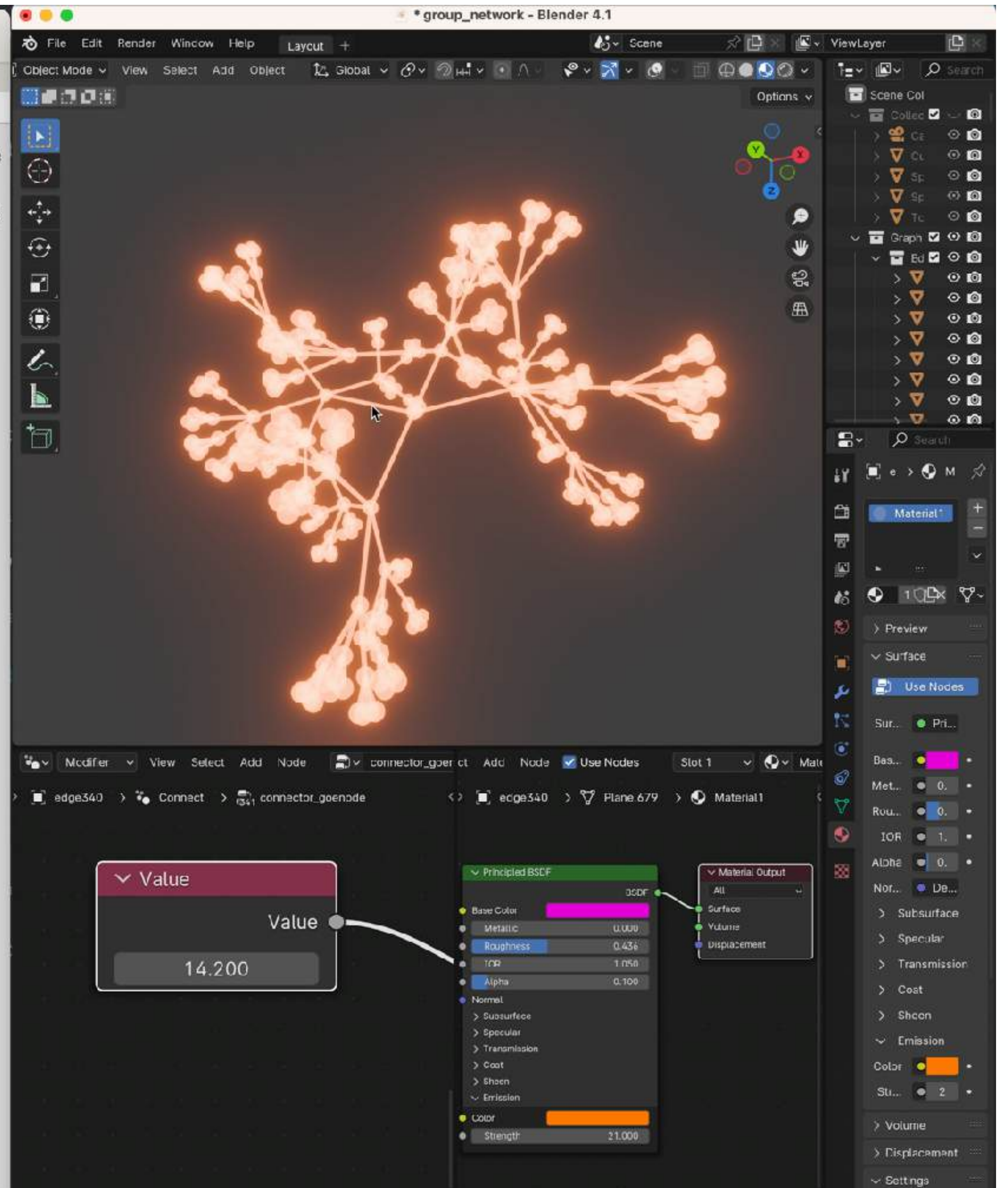
[ ]: G = nx.balanced_tree(6, 2)
node_positions = nx.spring_layout(G, dim=3, scale=1.9)
edges = list(G.edges)
draw_network(node_positions, edges, sphere_radius = 0.1)

[ ]: from IPython.display import display, Image

bpy.context.scene.render.resolution_x = 1000
bpy.context.scene.render.resolution_y = 1000

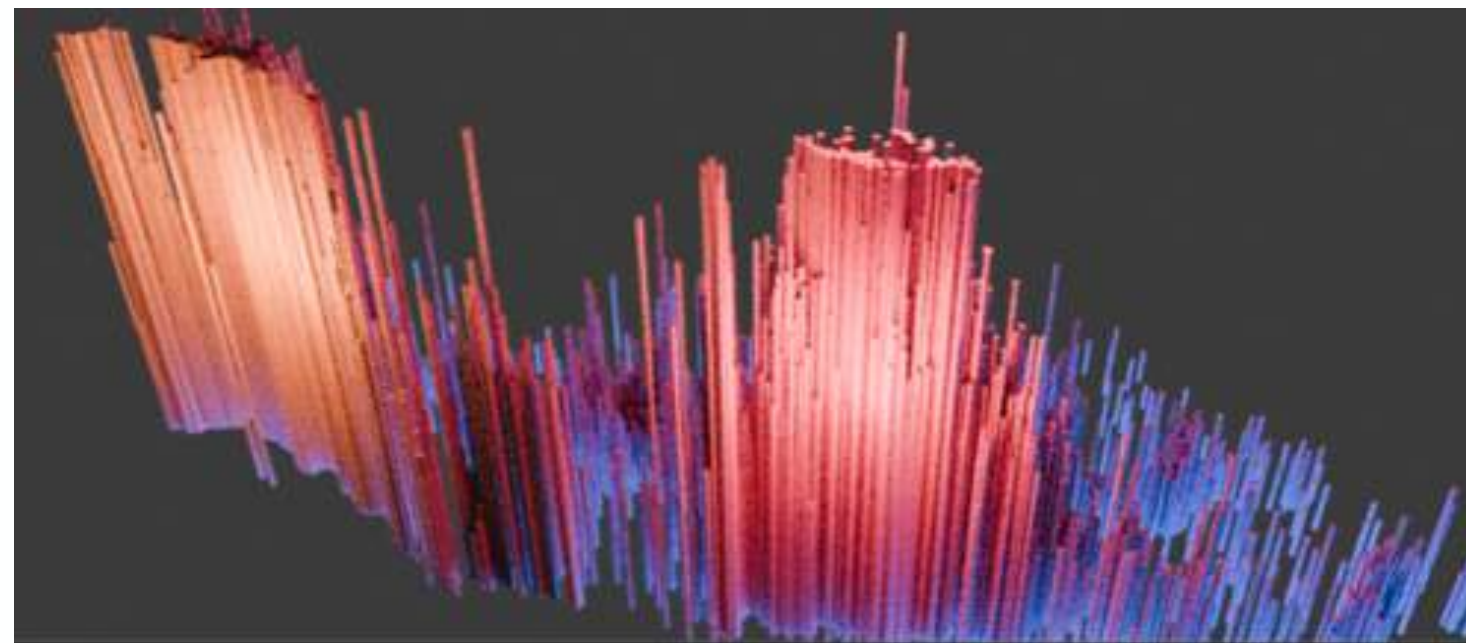
path = "/tmp/test.png"
bpy.context.scene.render.filepath = path
bpy.ops.render.render(write_still=True)

display(Image(filename=path , width=400))
```

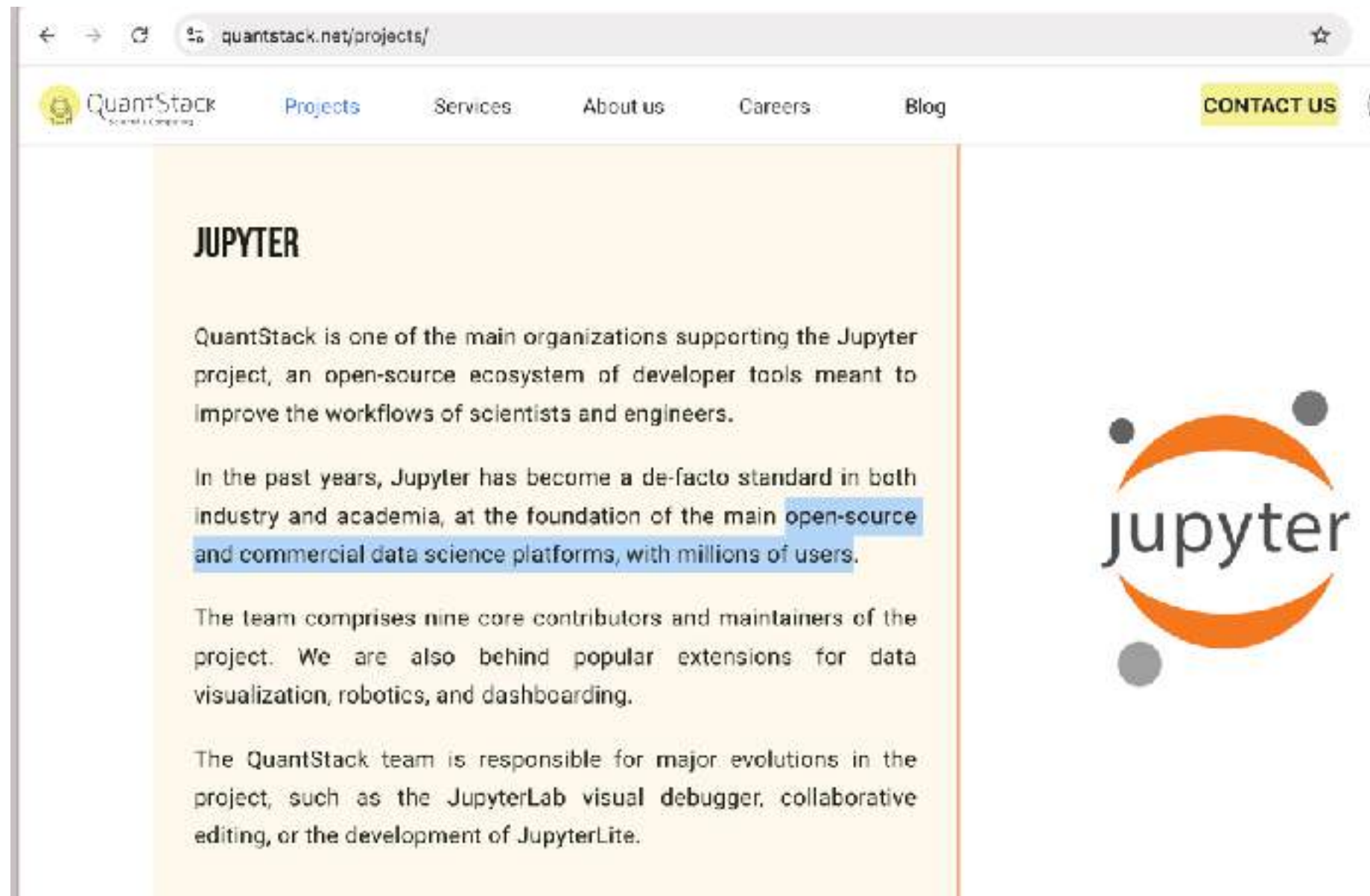


# Takeaway

- Setup the connection
- Interact with objects
- Data processing pipeline
- Time series
- Links to resources



# Users in Numbers




The screenshot shows the website [quantstack.net/projects/](https://quantstack.net/projects/). The page features a navigation bar with links for Projects, Services, About us, Careers, and Blog, along with a CONTACT US button. The main content area is titled "JUPYTER" and contains the following text:

QuantStack is one of the main organizations supporting the Jupyter project, an open-source ecosystem of developer tools meant to improve the workflows of scientists and engineers.

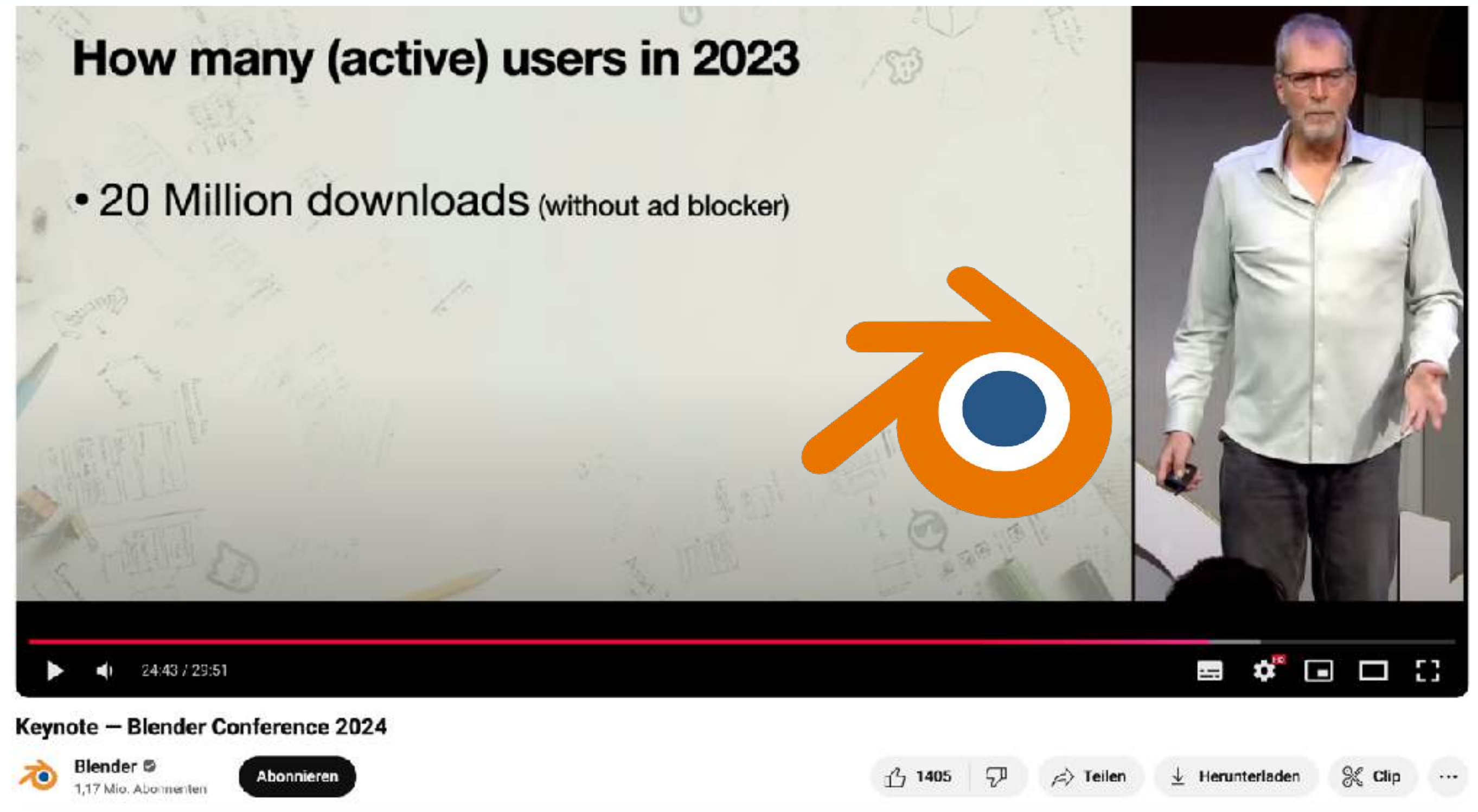
In the past years, Jupyter has become a de-facto standard in both industry and academia, at the foundation of the main open-source and commercial data science platforms, with millions of users.

The team comprises nine core contributors and maintainers of the project. We are also behind popular extensions for data visualization, robotics, and dashboarding.

The QuantStack team is responsible for major evolutions in the project, such as the JupyterLab visual debugger, collaborative editing, or the development of JupyterLite.




<https://quantstack.net/projects/>



The video player shows a keynote presentation titled "Keynote – Blender Conference 2024" by Blender, which has 1,17 Mio. Abonnenten. The video content includes the following text:

## How many (active) users in 2023

- 20 Million downloads (without ad blocker)



The video player interface shows a progress bar at 24:43 / 29:51 and various control icons. The channel name "Blender" and subscriber count "1,17 Mio. Abonnenten" are visible below the video.

# Python notebook

The image shows a Jupyter Notebook interface. At the top, the Jupyter logo and the text "Running Code Last Checkpoint: 10 months ago" are visible. Below this is a menu bar with "File", "Edit", "View", "Run", "Kernel", "Settings", and "Help". A toolbar contains icons for file operations and execution. The notebook content includes a title "Running Code", an introductory paragraph, a section "Code cells allow you to enter and run code" with a code cell containing `a = 10` and `print(a)`, a list of keyboard shortcuts, a section "Managing the Kernel" with a code cell containing `import time` and `time.sleep(10)`, and a final paragraph about kernel management.

jupyter Running Code Last Checkpoint: 10 months ago


File Edit View Run Kernel Settings Help

Interface Python 3 (ipykernel)

## Running Code

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefore runs Python code.

### Code cells allow you to enter and run code

Run a code cell using `Shift-Enter` or pressing the  button in the toolbar above:

```
[1]: a = 10
```


```
[2]: print(a)
```

10

There are two other keyboard shortcuts for running code:

- `Alt-Enter` runs the current cell and inserts a new one below.
- `Ctrl-Enter` run the current cell and enters command mode.

### Managing the Kernel

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted. Try running the following cell and then hit the  button in the toolbar above.

```
[3]: import time
```


```
time.sleep(10)
```

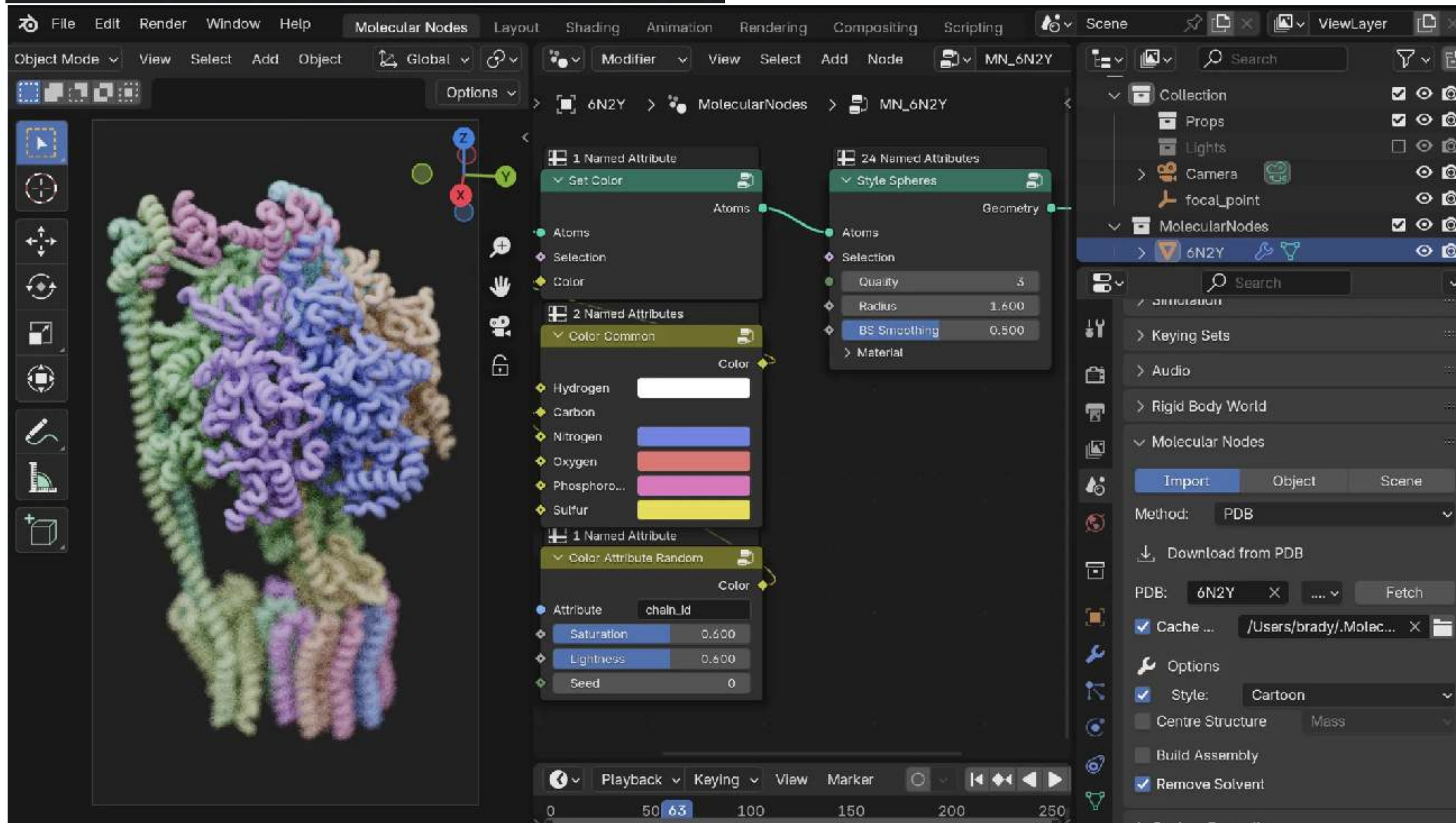
If the Kernel dies you will be prompted to restart it. Here we call the low-level system `libc.time` routine with the wrong argument via `ctypes` to segfault the Python interpreter:

# Blender in 100 seconds



# Blender extensions

 **Molecular Nodes**  
A toolbox for molecular import and animation in Blender.  
Add-on by bradyjohnston





# Blender extensions



## Molecular Nodes

A toolbox for molecular import and animation in Blender.

Add-on by bradyjohnston

<https://bradyjohnston.github.io/MolecularNodes/examples/>



14:41 / 27:45

Where the Light Touches Your Eyes | Phototransduction and Rhodopsi



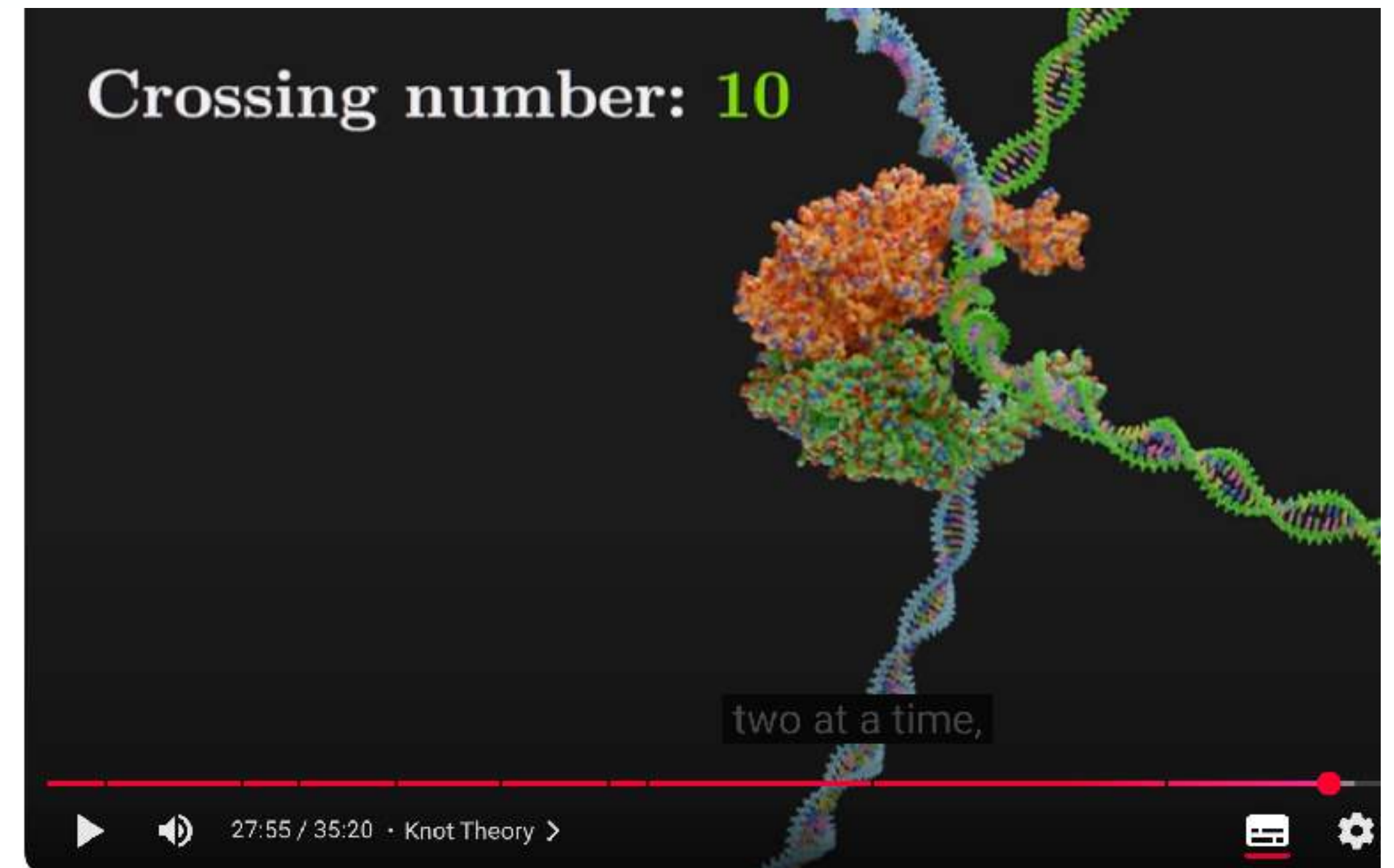
Clockwork

44.5K subscribers

Join

Subscribe

4.6K



Crossing number: 10

two at a time,

27:55 / 35:20 · Knot Theory

The Insane Math Of Knot Theory



Veritasium

17M subscribers

Subscribe

207K



Share

8M views 1 year ago

# Blender extensions



## Bioxel Nodes

For scientific volumetric data visualization in Blender.

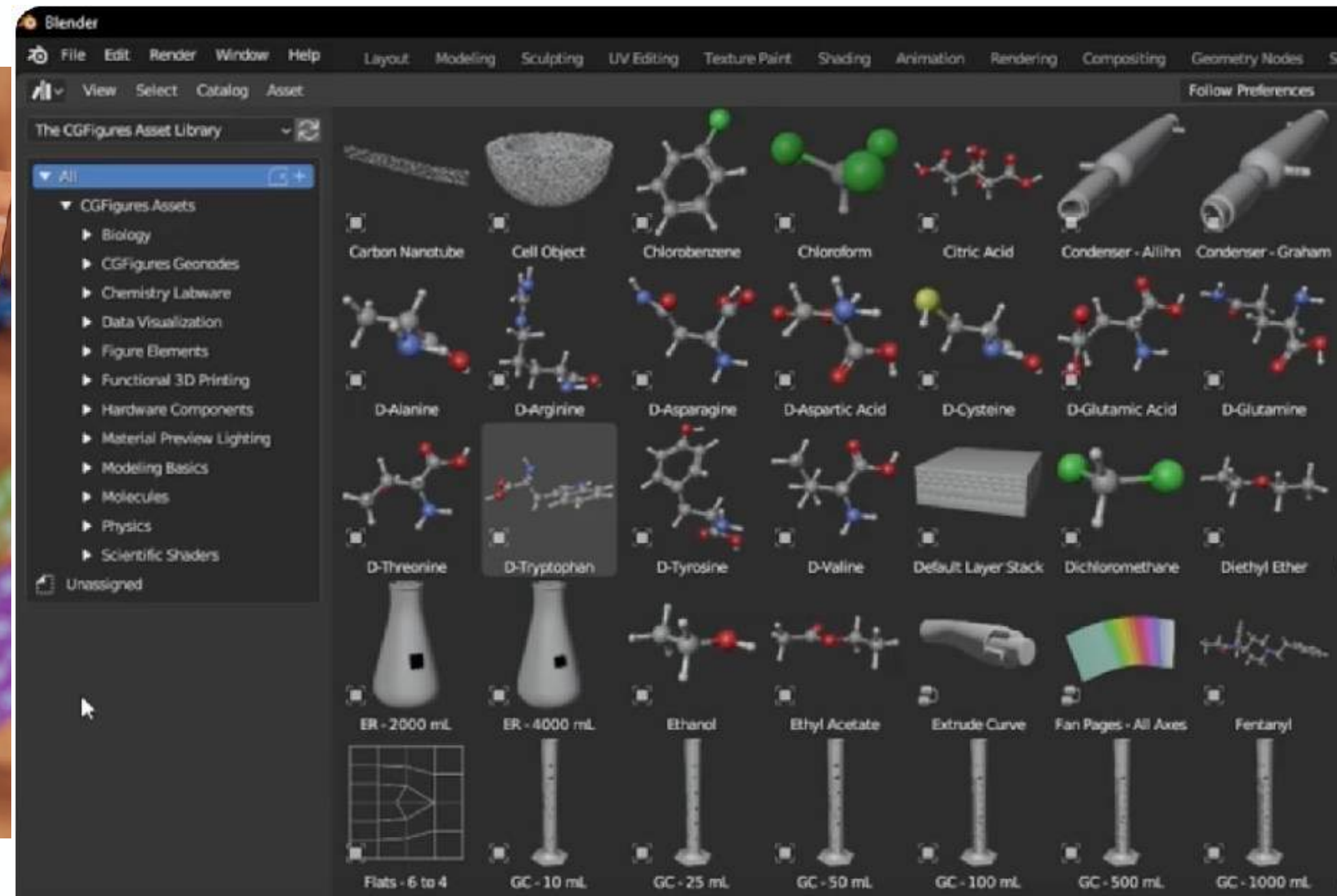
Add-on by icdr



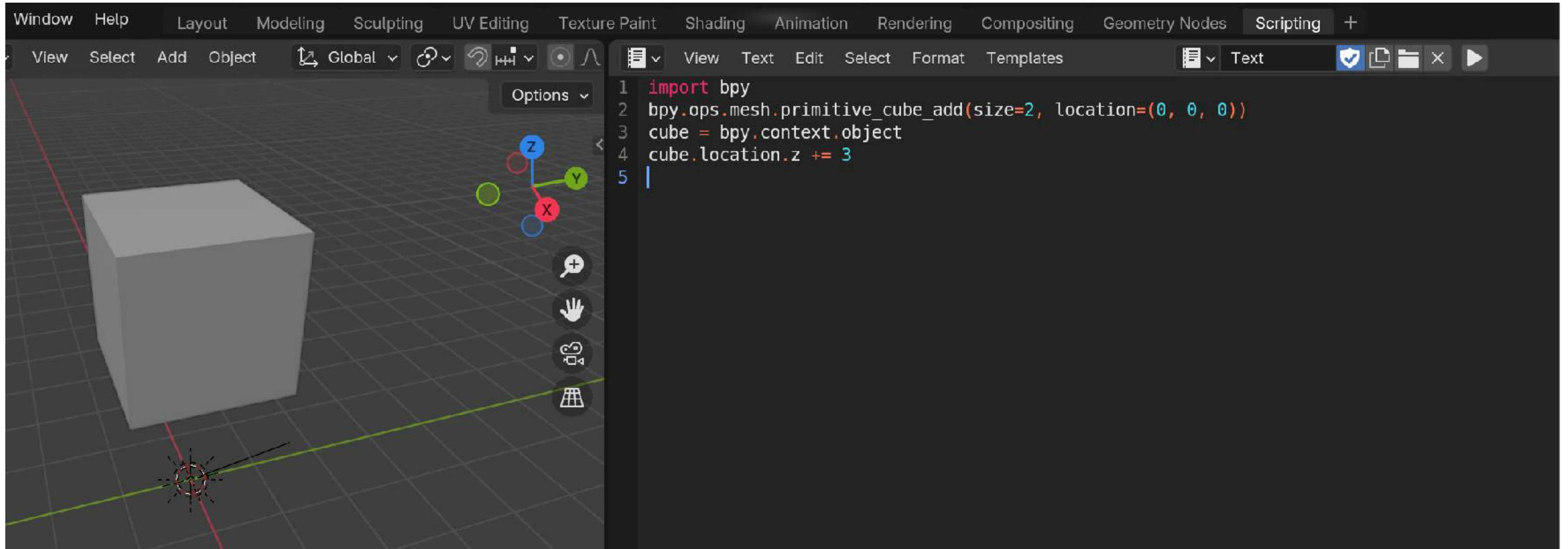
# Blender extensions

## free resources for scientific visualization

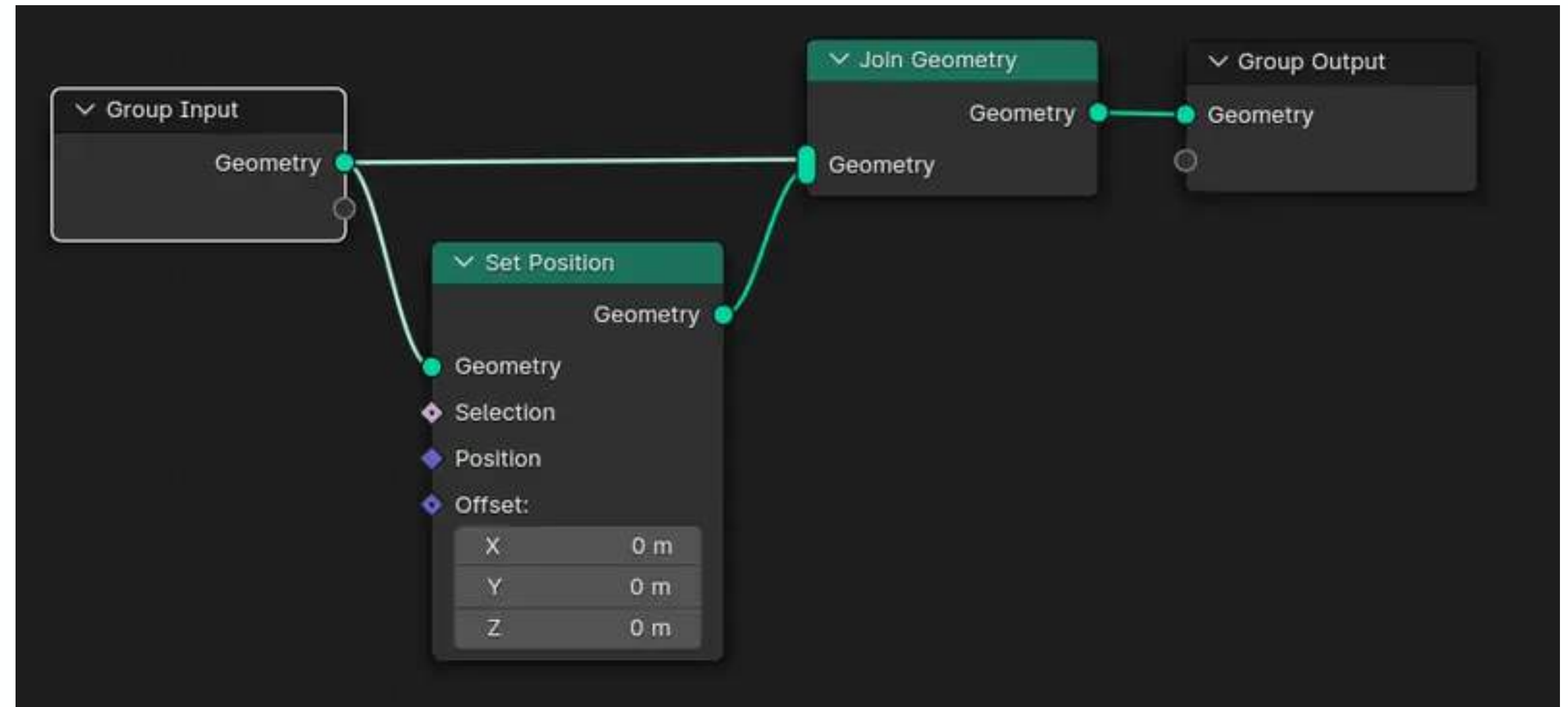
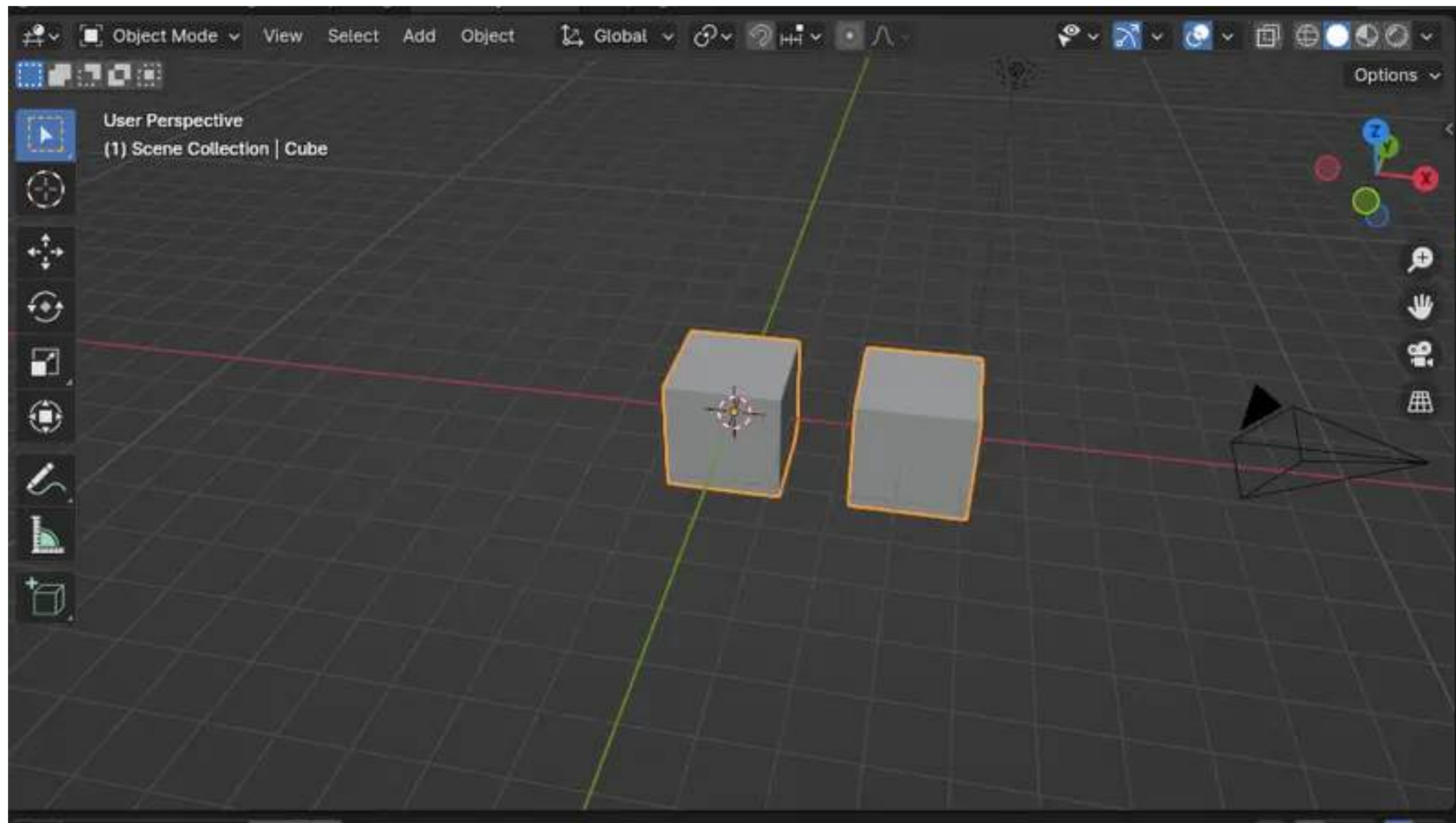
<https://www.cgfigures.ca/assetlibrary>



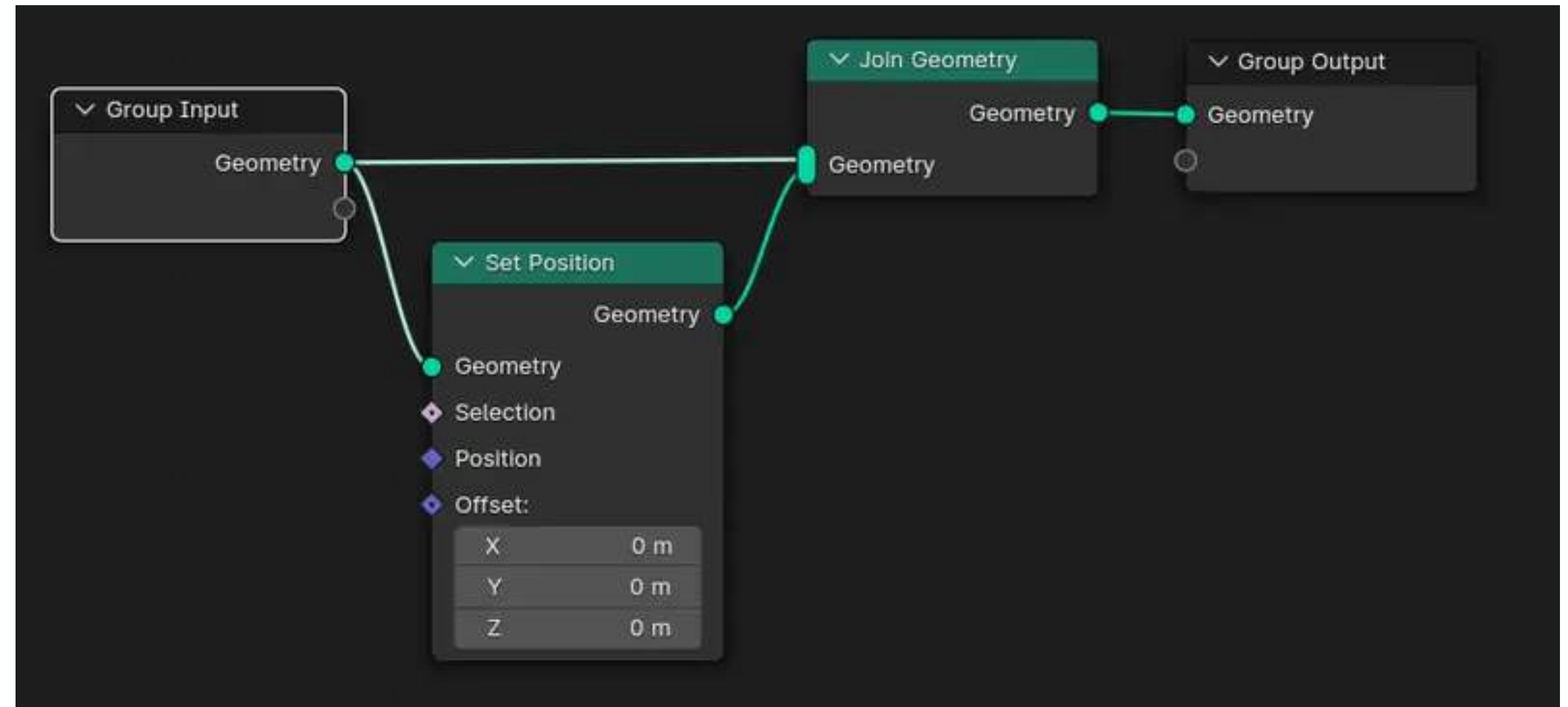
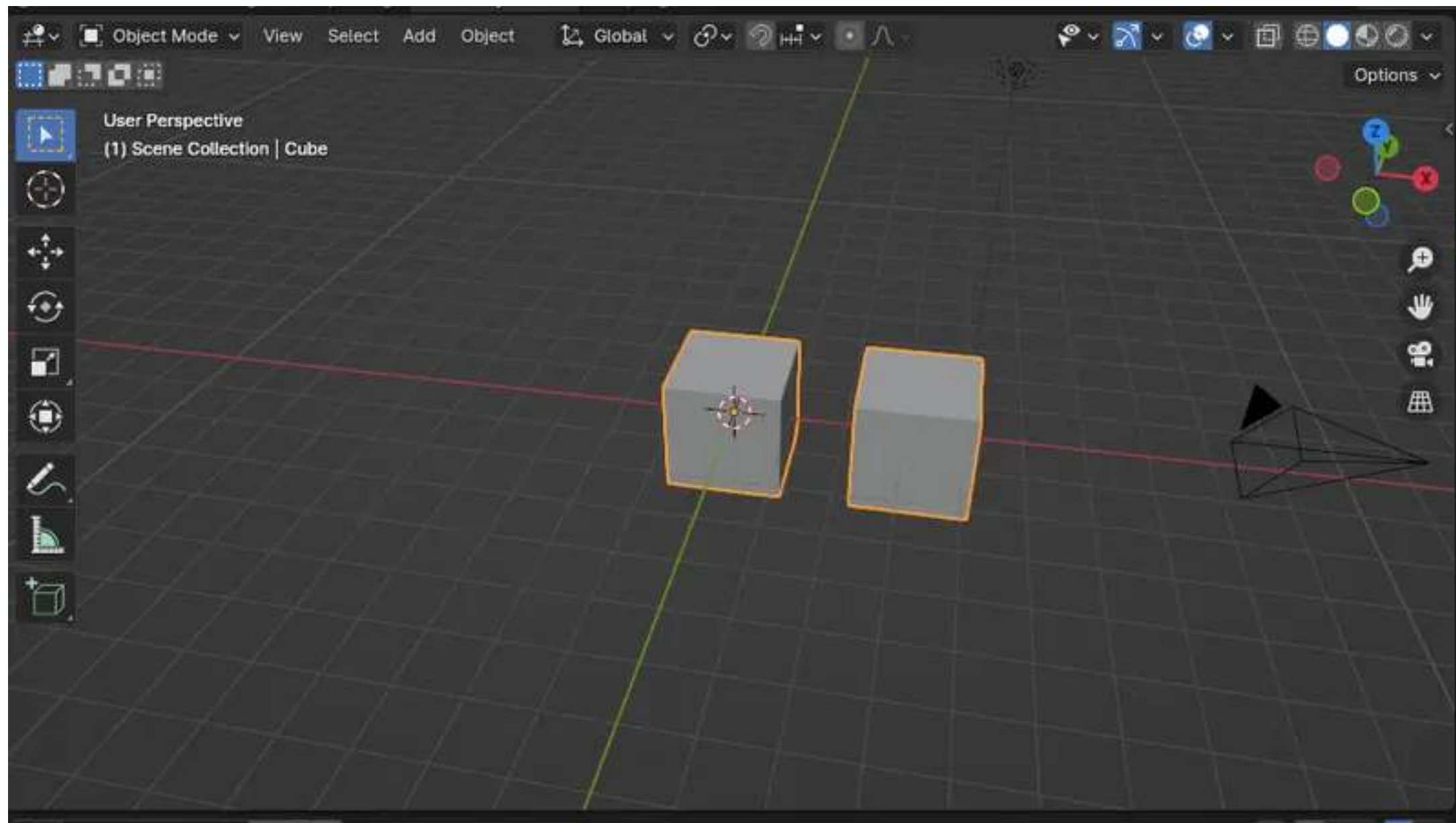
# Blender Scripting



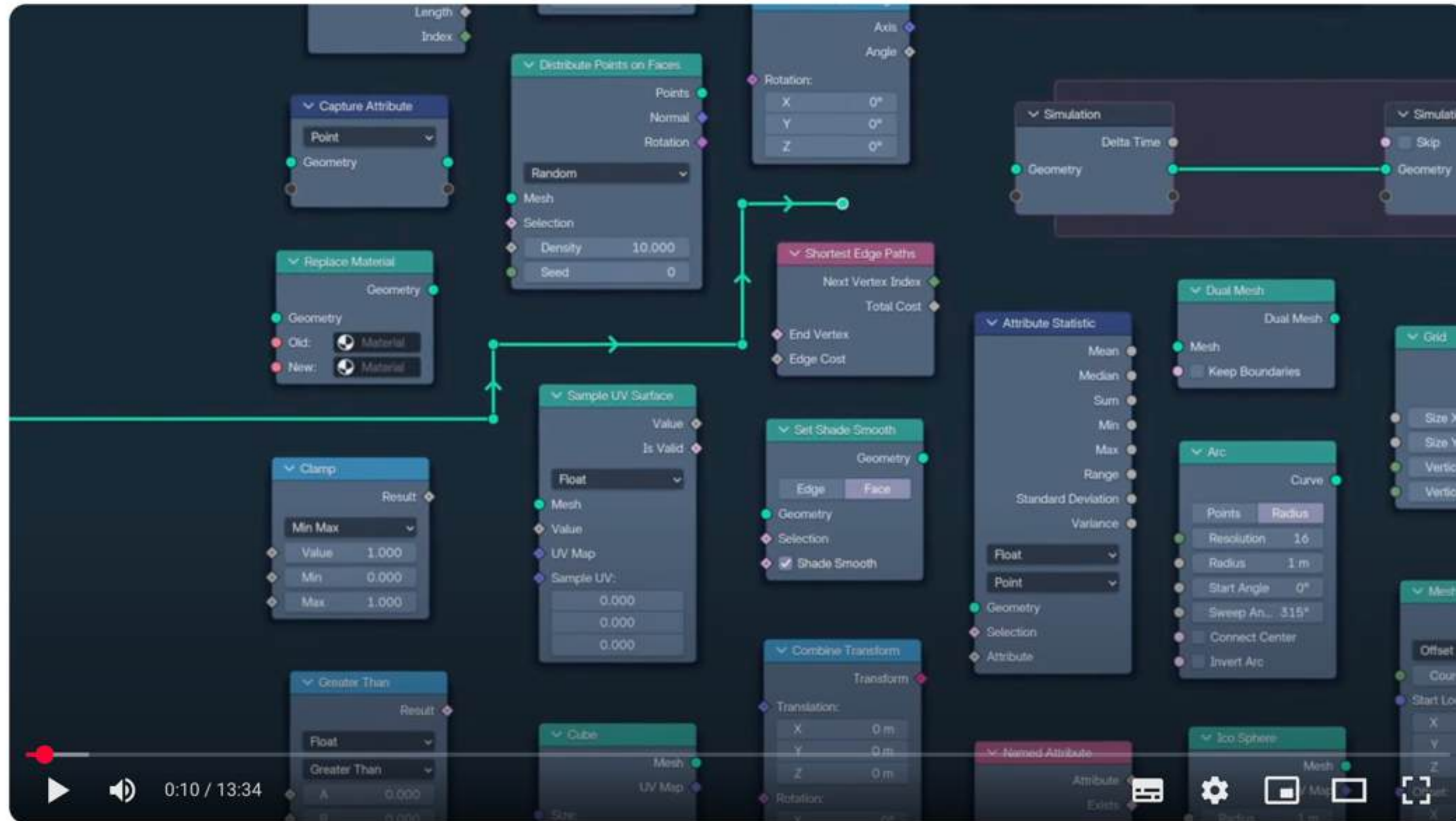
# Blender Geometry Nodes



# Blender Geometry Nodes




# Blender Geometry Nodes Tutorial




A guide to attributes & fields - blender geometry nodes

 harry blends  
14.5K subscribers

Subscribe

 6.8K



 Share

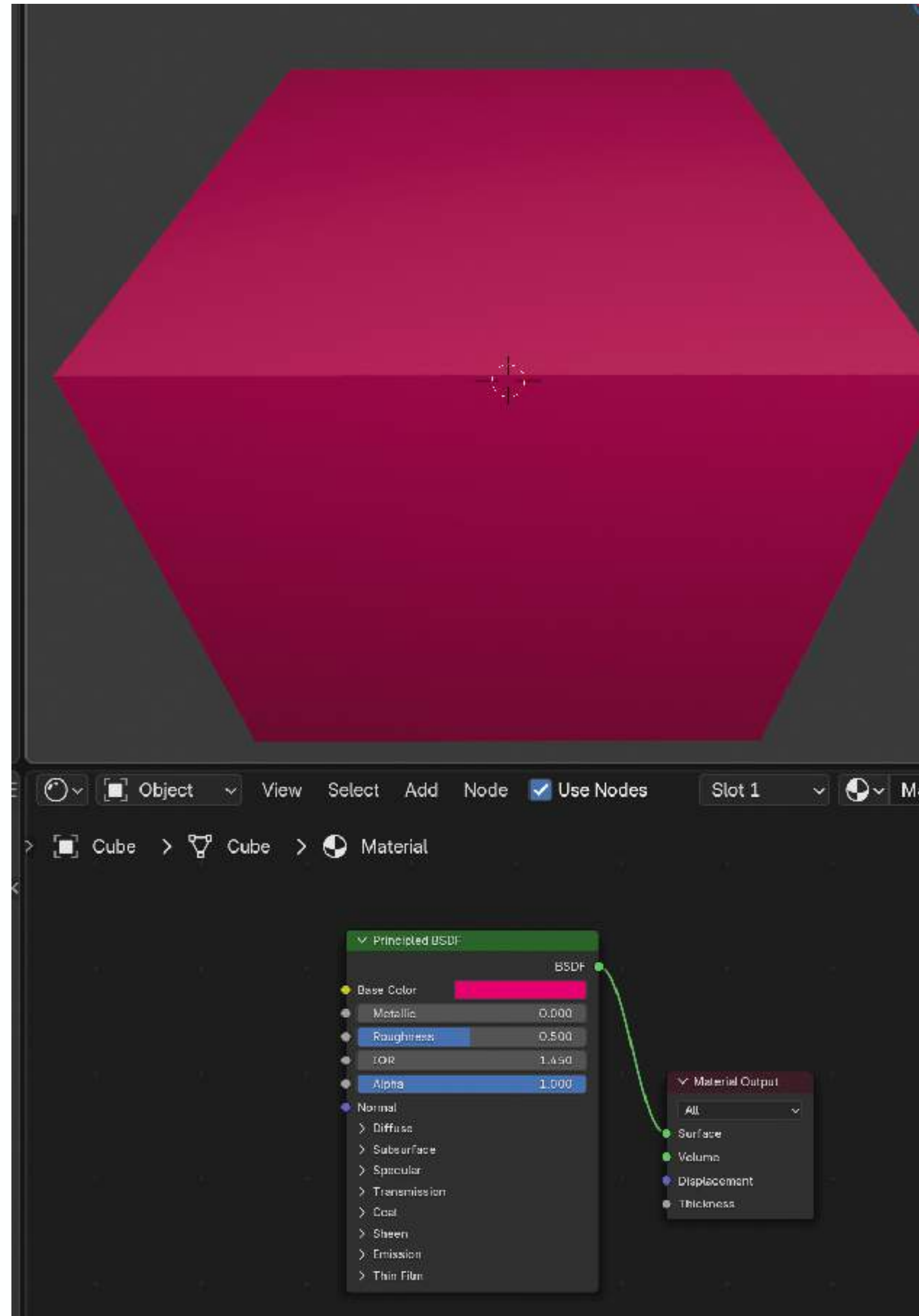
 Download



76K views 5 months ago #blenderanimation #blender3d #b3d

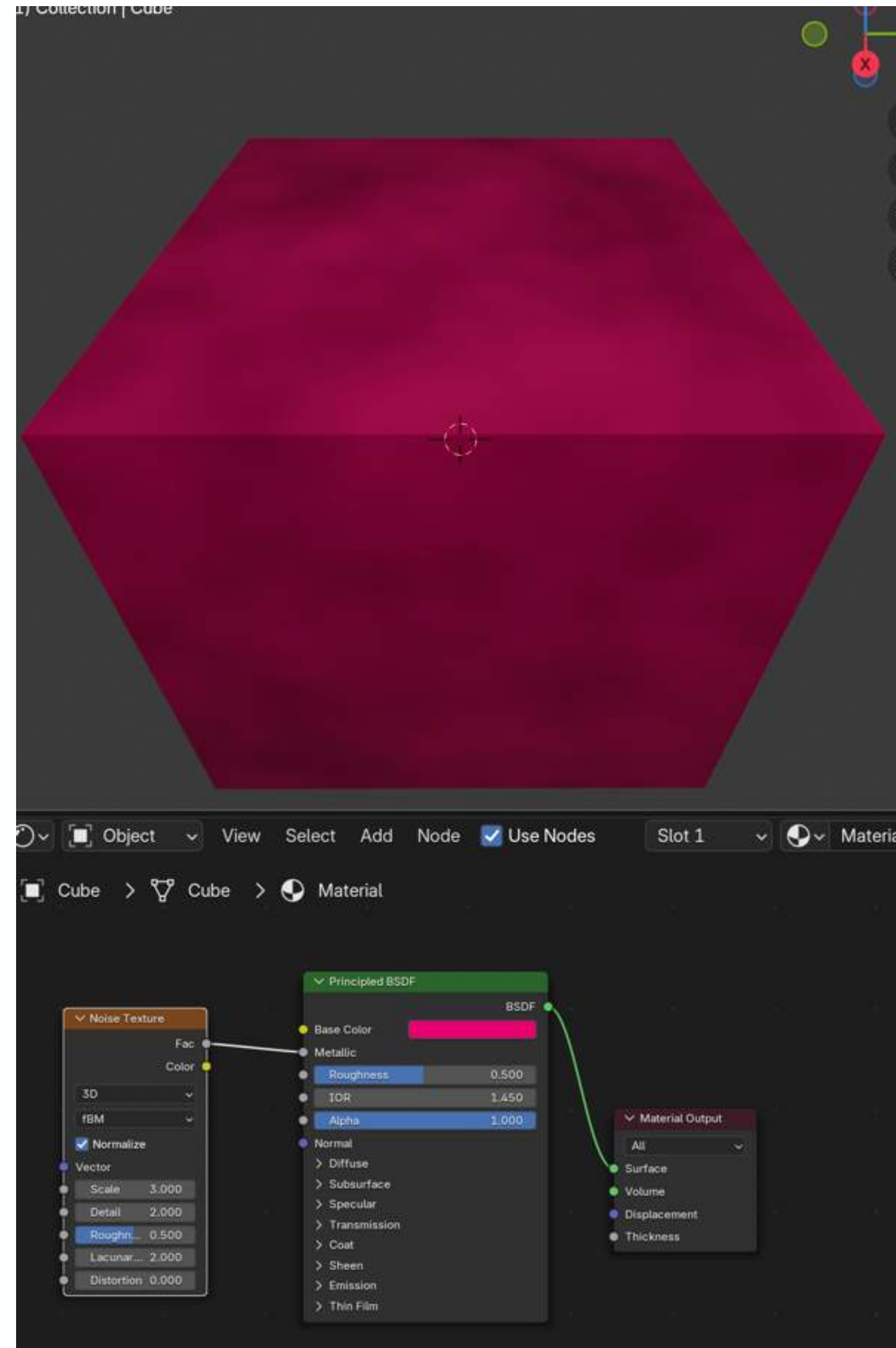
harry blends - <https://www.youtube.com/watch?v=a-4oCHe-hDE&t=61s>

# Shader Nodes

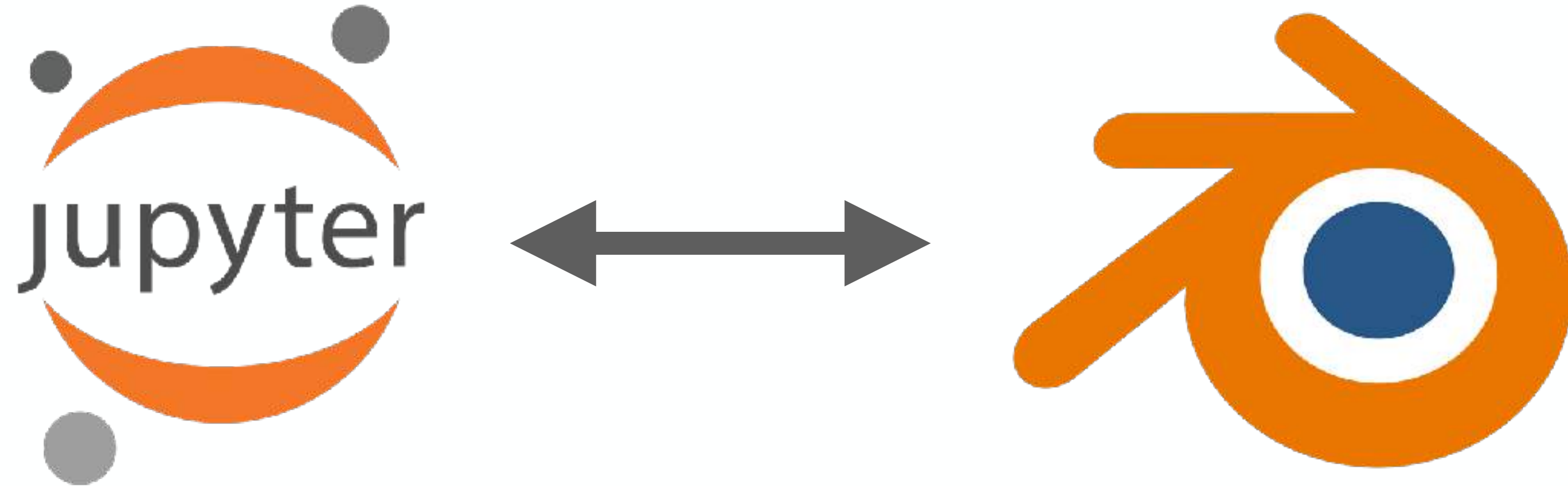




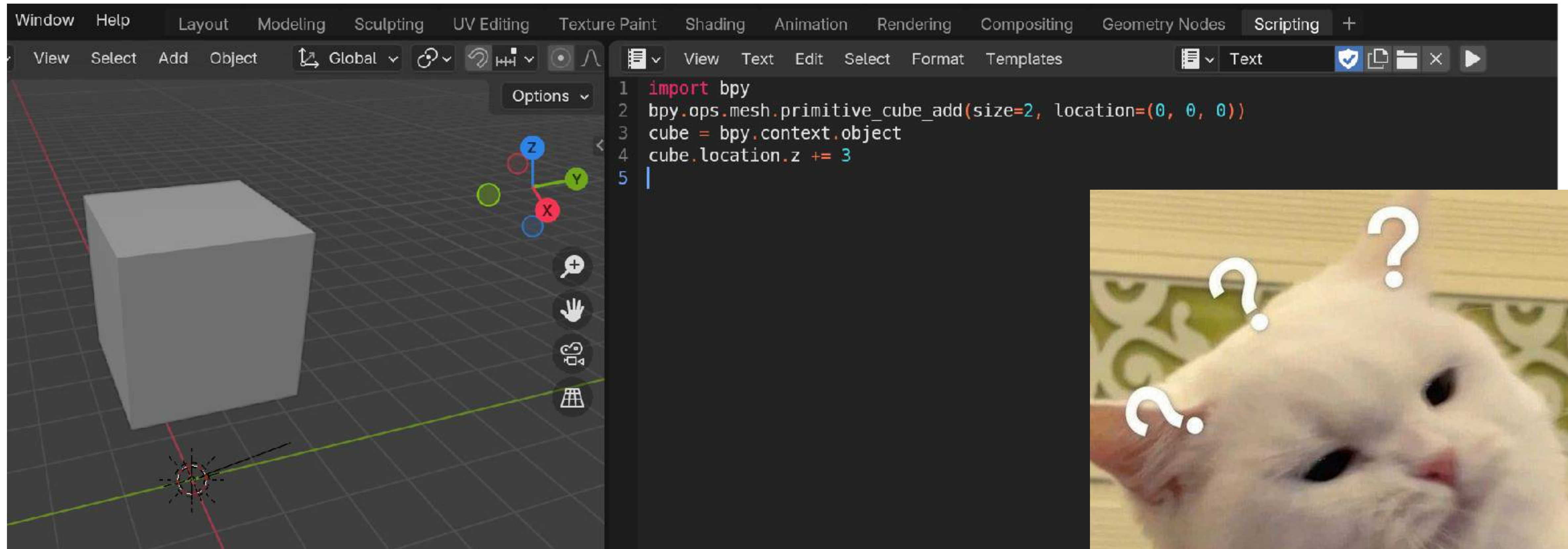
# Shader Nodes



# Blender and Notebooks



# Blender Python editor



# 1. Iterative Workflows

The image displays two side-by-side windows illustrating an iterative workflow. The left window is a Jupyter Notebook titled "teaser1.ipynb" containing the following Python code:

```
import bpy
bpy.ops.mesh.primitive_plane_add(size=3)
plane = bpy.context.object

plane.location.z += 1

plane.location.z -= 2

len(plane.data.vertices)

vertices = [(v.co.x, v.co.y, v.co.z) for v in plane.data.vertices]
import pandas as pd
df = pd.DataFrame(vertices, columns=['X', 'Y', 'Z'])
df
```

The right window is the Blender 4.2.3 LTS interface. It shows a 3D perspective view of a scene with a plane object. The interface includes a top menu bar (File, Edit, Render, Window, Help), a toolbar on the left, and a right-hand panel with various toolbars and panels. The scene is currently in Object Mode, and the plane is selected. The right-hand panel shows the Properties panel for the selected object, with various settings and options visible.

```
localhost
lab - JupyterLab
Circular Tree — NetworkX 3.3 documentation

File Edit View Run Kernel Tabs Settings Help

multiple_networks.ipynb
Code blender

[2]: import networkx as nx
import bpy

G = nx.balanced_tree(4, 4)
node_positions = nx.spring_layout(G, dim=3, scale=1.9)
edges = list(G.edges)
draw_network(node_positions, edges, sphere_radius = 0.05)

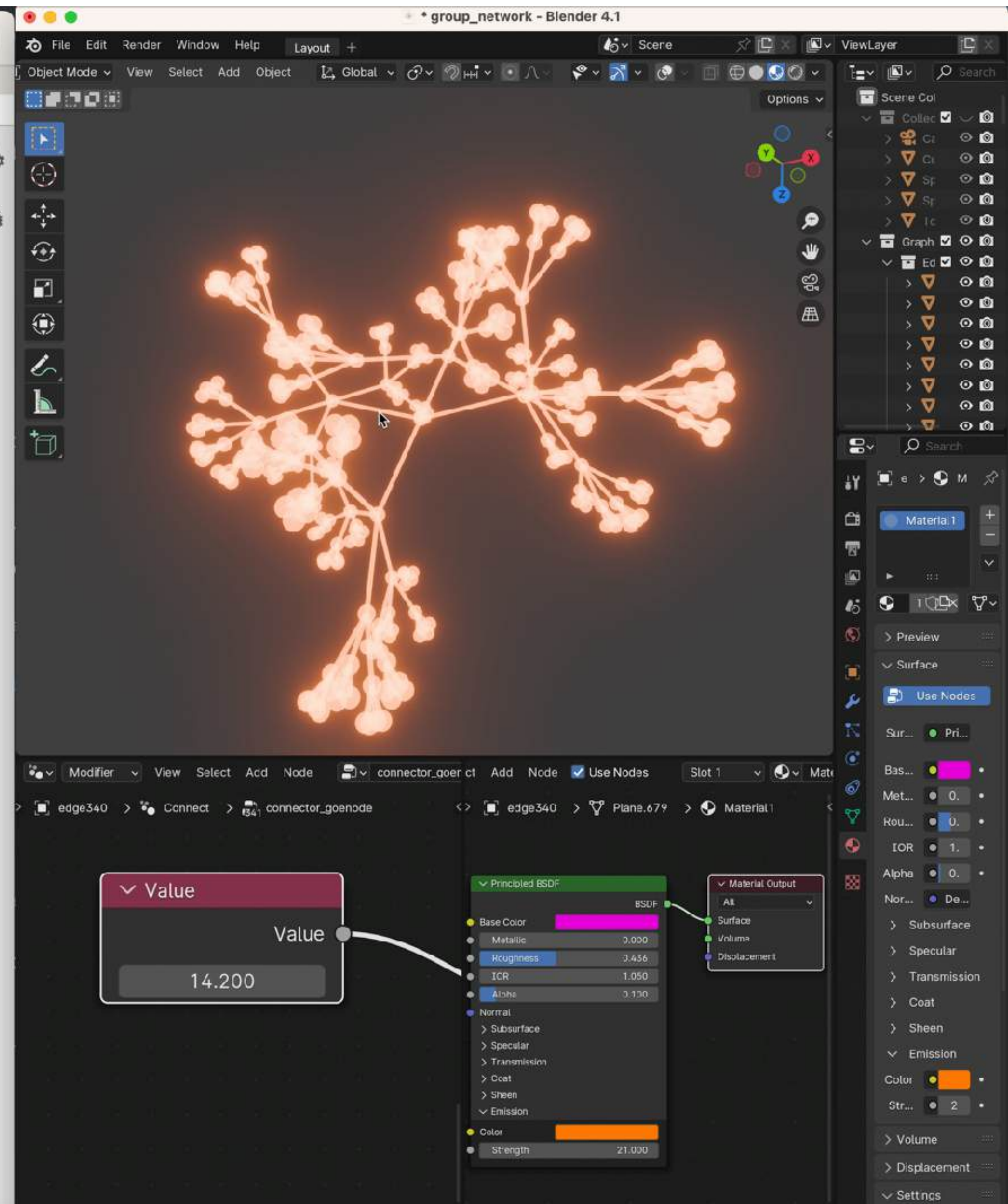
[ ]: G = nx.balanced_tree(6, 2)
node_positions = nx.spring_layout(G, dim=3, scale=1.9)
edges = list(G.edges)
draw_network(node_positions, edges, sphere_radius = 0.1)

[ ]: from IPython.display import display, Image

bpy.context.scene.render.resolution_x = 1000
bpy.context.scene.render.resolution_y = 1000

path = "/tmp/test.png"
bpy.context.scene.render.filepath = path
bpy.ops.render.render(write_still=True)

display(Image(filename=path, width=400))
```



```
n3c_single_mesh_netwokx.ipynb  blender
```

```
netw = bpy.data.objects.get('GraphObject')
netw_modif = netw.modifiers["GeometryNodes"]
netw_modif["Socket_2"] = 1.0
bpy.context.object.data.update()
```

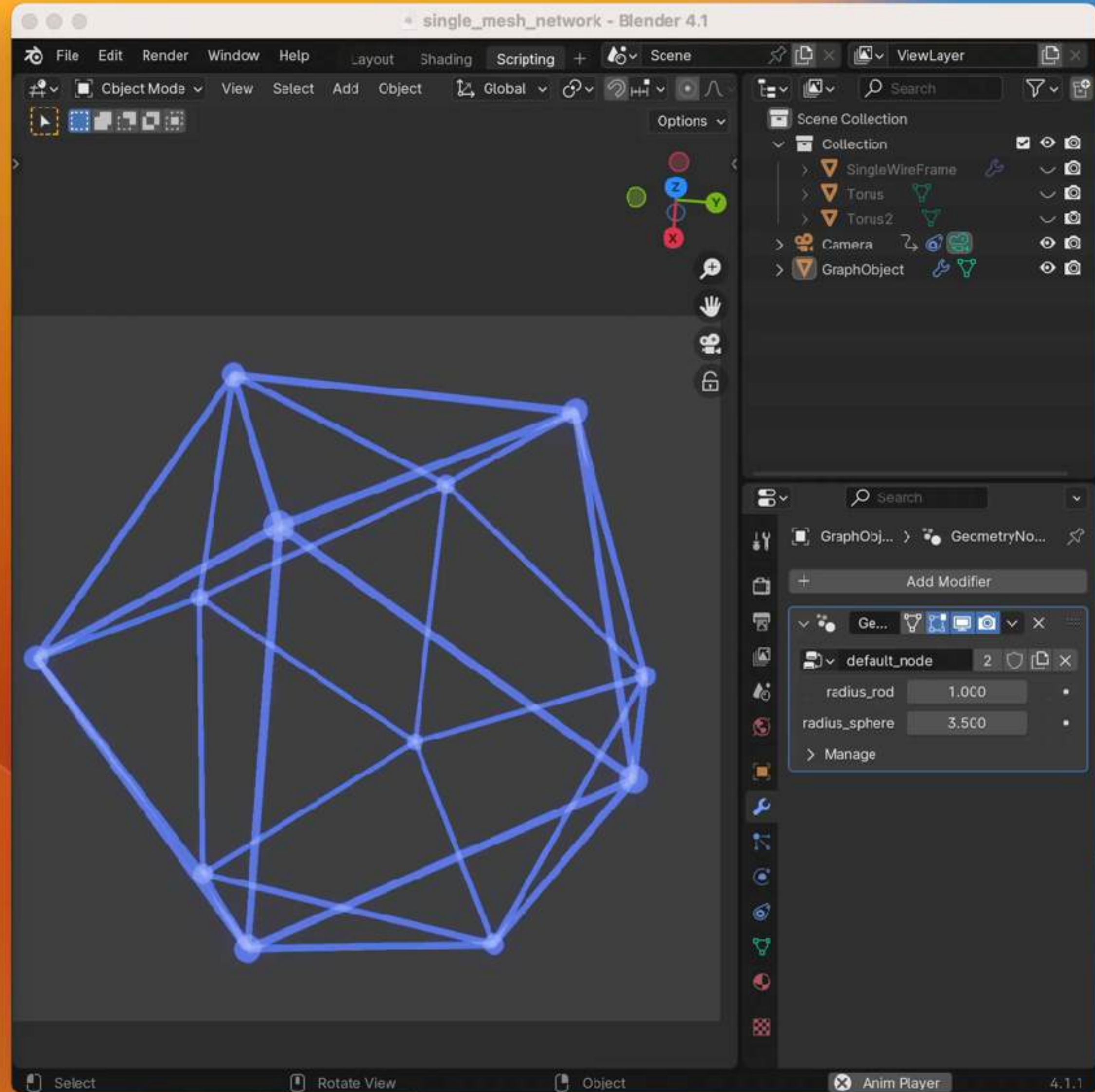
```
netw_modif["Socket_2"] = 9.5
bpy.context.object.data.update()
```

```
material2 = bpy.data.materials.get("Material2")
for node in netw_modif.node_group.nodes:
    if node.type == 'SET_MATERIAL':
        node.inputs['Material'].default_value = material2
```

```
netw_modif["Socket_3"] = 22.0
bpy.context.object.data.update()
```

```
material1 = bpy.data.materials.get("Material1")
for node in netw_modif.node_group.nodes:
    if node.type == 'SET_MATERIAL':
        node.inputs['Material'].default_value = material1
```

```
netw_modif["Socket_2"] = 2.0
netw_modif["Socket_3"] = 3.5
bpy.context.object.data.update()
```



```
n3d_color_edge_to_face_brady.ipynb  blender
```

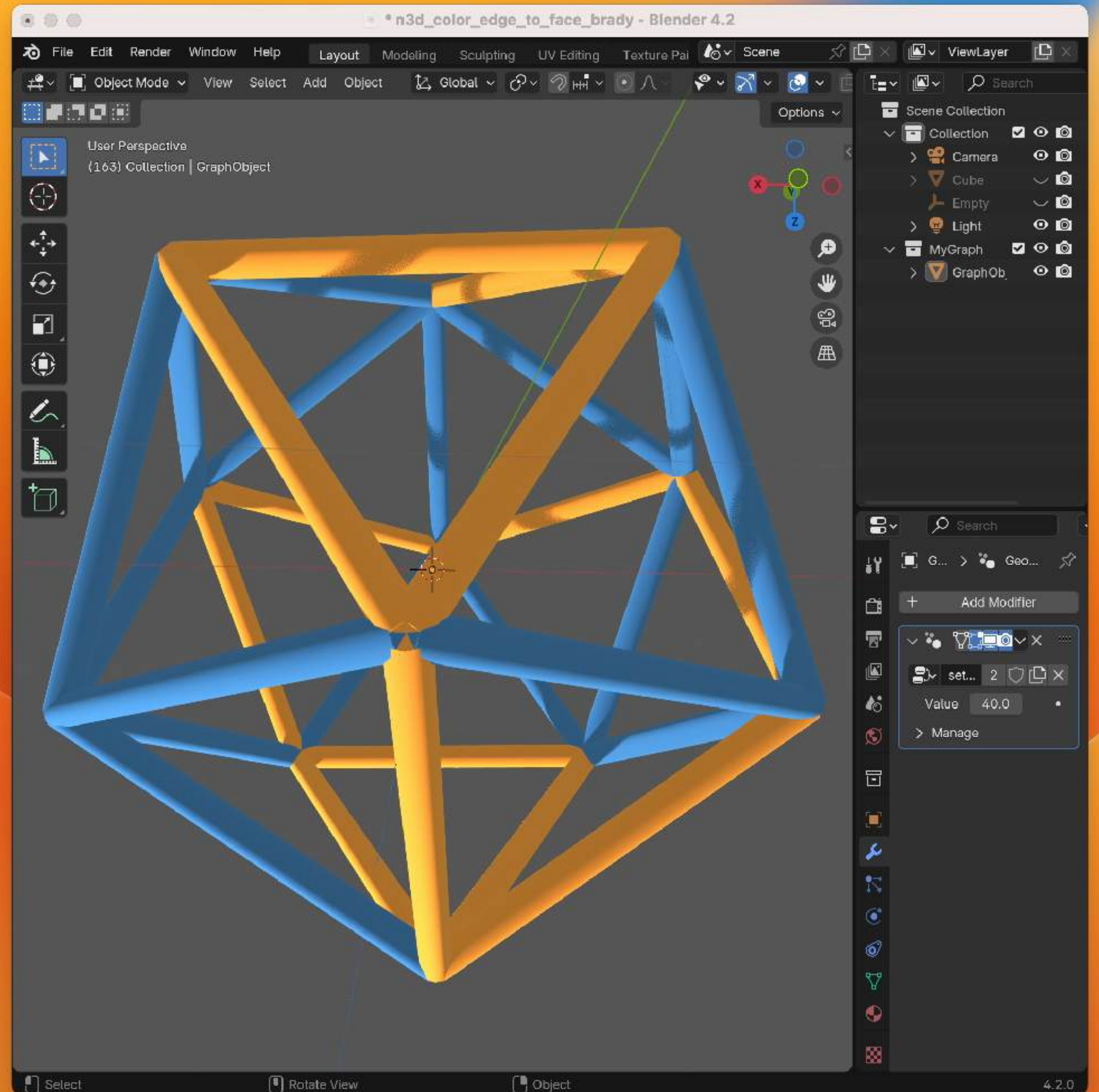
```
color = (1.0, 1.0, 1.0) # WHITE
for edge in obj.data.edges:
    v1, v2 = edge.vertices
    set_edge_color(obj, (v1, v2), color)
```

```
38 color = (1.0, 0.0, 0.0) # RED
    set_edge_color(obj, (0, 1), color)
```

```
46 colors = [
    (1.0, 0.588, 0.149), # ORANGE
    (0.176, 0.431, 0.686) # BLUE
]
for edge in obj.data.edges:
    v1, v2 = edge.vertices
    color = random.choice(colors)
    set_edge_color(obj, (v1, v2), color)
```

```
40 color = (1.0, 1.0, 1.0) # WHITE
for edge in obj.data.edges:
    v1, v2 = edge.vertices
    set_edge_color(obj, (v1, v2), color)
```

```
41 my_edges = [(0, 1), (0, 5), (0, 7), (0, 8), (0, 11)]
color = (0.0, 0.0, 1.0) # BLUE
for edge in my_edges:
    set_edge_color(obj, edge, (0.0, 0.0, 1.0))
```



[1]:

```
import bpy  
  
bpy.ops.wm.open_mainfile(filepath='my_donut.blend')
```

Read blend: "/Users/jan-hendrik/projects/ipyblender-gui/my\_donut.blend"

[1]:

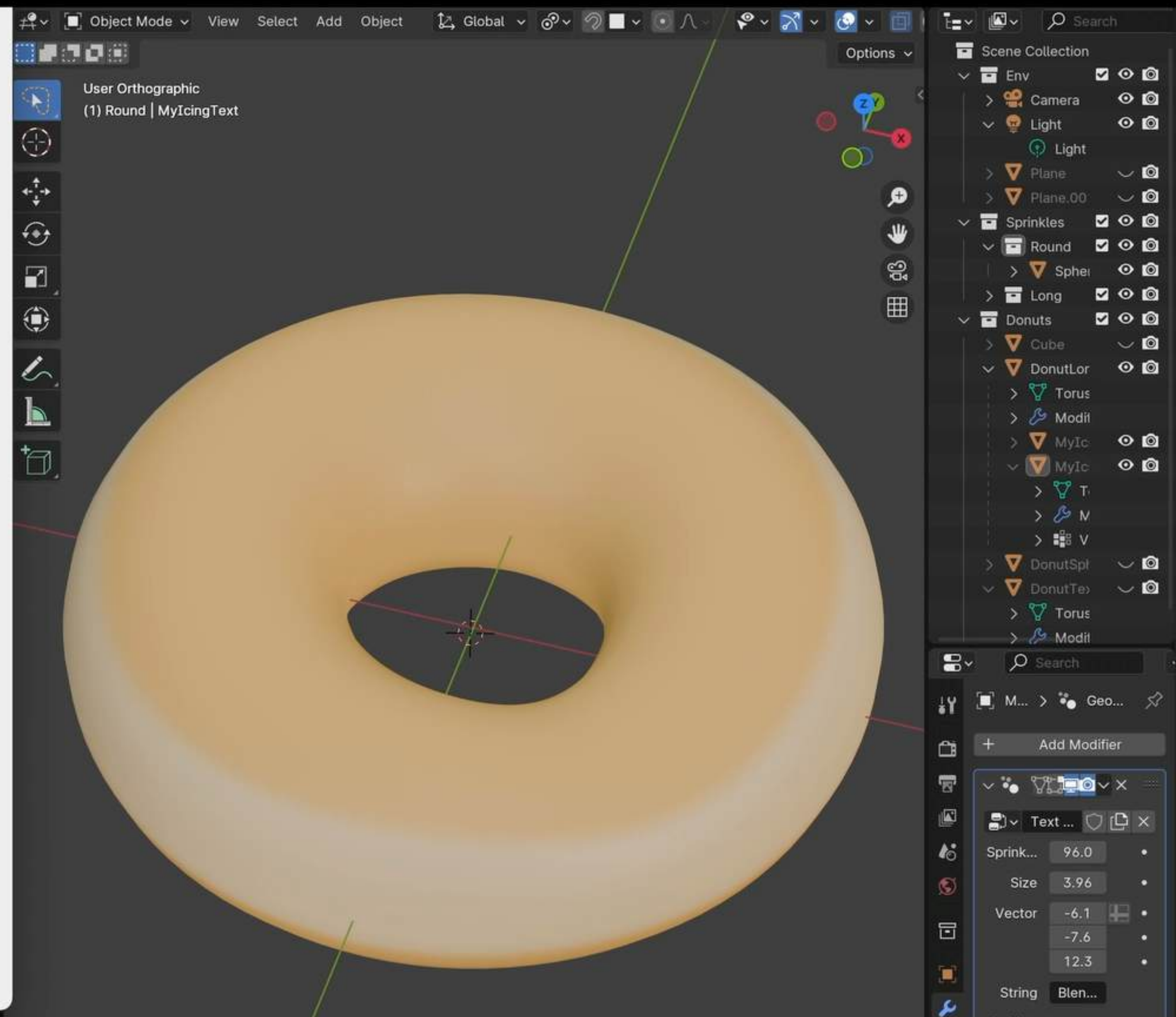
{'FINISHED'}

[2]:

```
def show(obj):  
    bpy.data.objects.get(obj).hide_viewport = False  
def hide(obj):  
    bpy.data.objects.get(obj).hide_viewport = True  
show("DonutLong")
```

[ ]:

```
show("MyIcing")
```





# 2. Blender & JavaScript



The image shows a JupyterLab environment with a Python notebook on the left and the Blender 2.80 interface on the right. The notebook contains the following code:

```
[1]: import bpy
import ipywidgets as widgets
from IPython.display import display, clear_output

slider = widgets.FloatSlider(value=0.0, min=0.0, max=1, step=0.01,
                             description='Rotation:', continuous_update=True)
display(slider)

[2]: def update_cube_rotation(rotation_value):
    cube = bpy.data.objects['Cube']
    cube.rotation_euler.z = rotation_value
    bpy.context.view_layer.update()

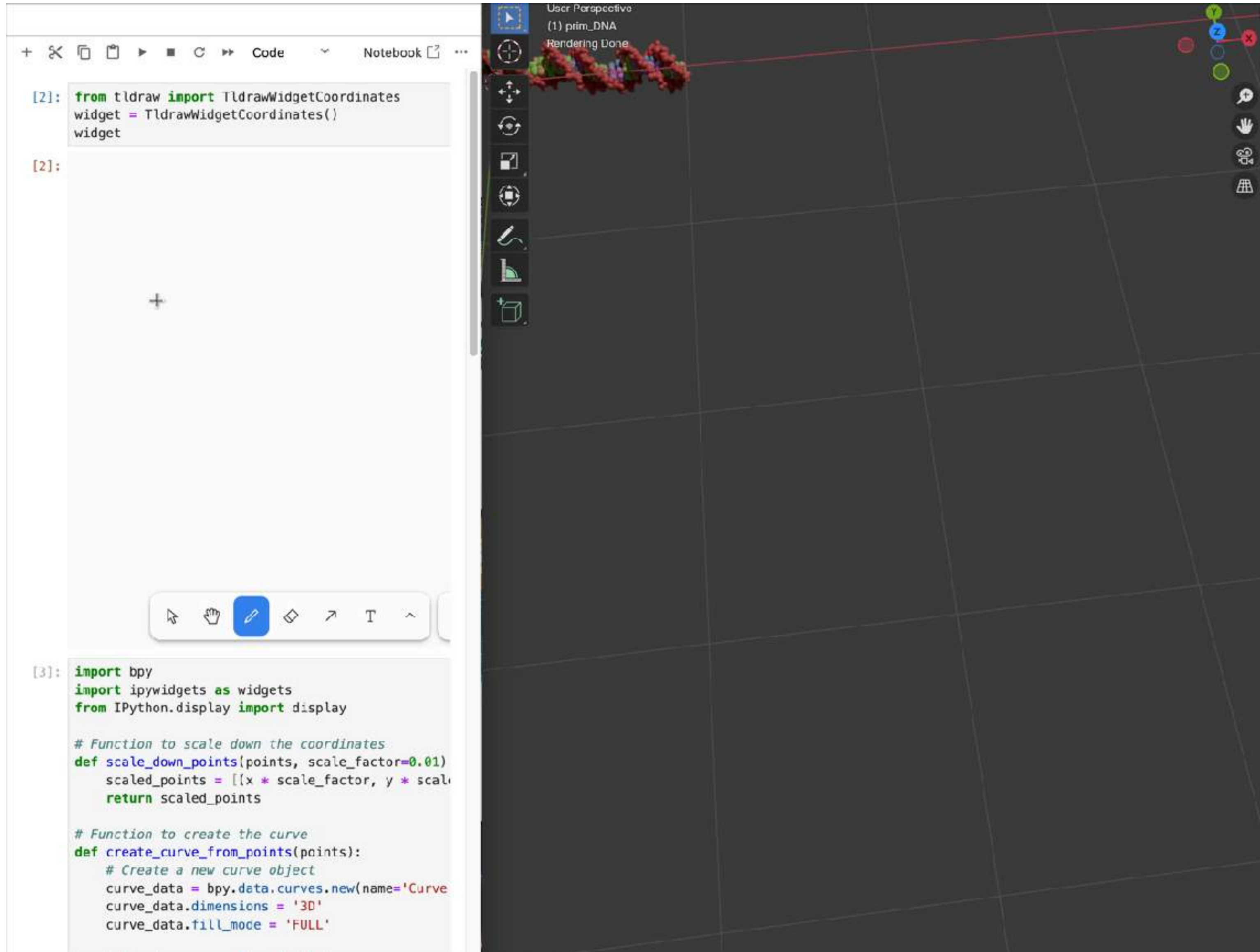
# Function to handle slider value changes
def on_slider_change(change):
    update_cube_rotation(change['new'])

# Observe the slider
slider.observe(on_slider_change, names='value')

# Initial update of the cube's rotation
update_cube_rotation(slider.value)
```

The Blender interface shows a 3D viewport with a cube in the center. The cube is highlighted with an orange outline. The right sidebar shows the Properties panel for the selected cube, with the Transform tab active. The rotation values for the X, Y, and Z axes are visible.

# 2.1. interactive Whiteboard JS → jupyter →



The screenshot displays a JupyterLab environment. On the left, a code editor shows two code cells. The first cell imports `TldrawWidgetCoordinates` from `tldraw`. The second cell imports `bpy` and `ipywidgets` from `IPython.display`, and defines two functions: `scale_down_points` for scaling coordinates and `create_curve_from_points` for creating a 3D curve. The right side of the interface shows a 3D visualization of a molecular structure with a red line, overlaid on a grid. A toolbar with various interaction tools is visible on the left side of the 3D view.

```
[2]: from tldraw import TldrawWidgetCoordinates
      widget = TldrawWidgetCoordinates()
      widget

[2]:

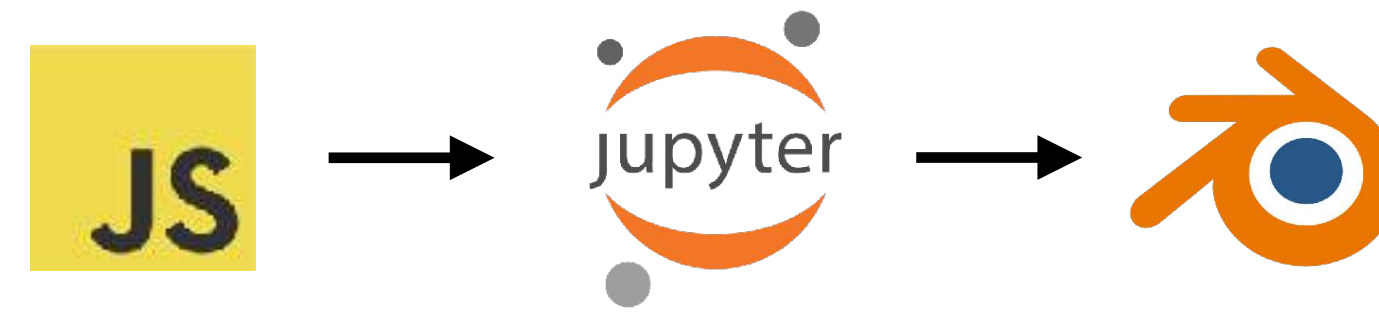
[3]: import bpy
      import ipywidgets as widgets
      from IPython.display import display

      # Function to scale down the coordinates
      def scale_down_points(points, scale_factor=0.01)
        scaled_points = [(x * scale_factor, y * scale_factor)]
        return scaled_points

      # Function to create the curve
      def create_curve_from_points(points):
        # Create a new curve object
        curve_data = bpy.data.curves.new(name='Curve')
        curve_data.dimensions = '3D'
        curve_data.fill_mode = 'FULL'
```

<https://github.com/kolibril13/jupyter-tldraw>

# 2.2. GPT Vision API



The screenshot displays a JupyterLab environment. On the left, a code editor window titled 'blender' contains the following Python code:

```
[1]: from tldraw import MakeReal
from api_key import api_key

custom_prompt = f"""
Use the bpy API to generate a blender scene.
In the provided image, you will see how the scene should look like.
Use 3D shape primitives to add the desired shapes and their colors.
Also add a camera and a light.

Reply ONLY with python code.
"""

m = MakeReal(width=650, height = 400, api_key = api_key,
             prompt = custom_prompt, run_next_cell = True)
m
```

Below the code editor is a toolbar with a 'Make Real' button and a color palette. The right side of the interface shows a 3D viewport in 'User Perspective' view, displaying a simple grey cube on a dark grid floor. The viewport includes a navigation toolbar on the left and a top status bar showing '(1) Collection | Cube'.

# 2.2. GPT Vision API



```
[14]: from tldraw import MakeReal
      from api_key import api_key

      custom_prompt = f"""

      In the provided image, you will see how a blender scene should look like.
      Please try to generate this scene in Blender.
      Use the bpy API to generate for that.
      Red text and red arrows will describe motion of the animation.
      Don't add the text and arrow itself.
      Instead, try to understand the motion that is described.
      Animate this motion using keyframes.
      in anycase USE keyframes please!
      Reply ONLY with python code.
      """

      m = MakeReal(width=650, height = 400, api_key = api_key, prompt = custom_prompt , run_next_
      m
```

[14]:

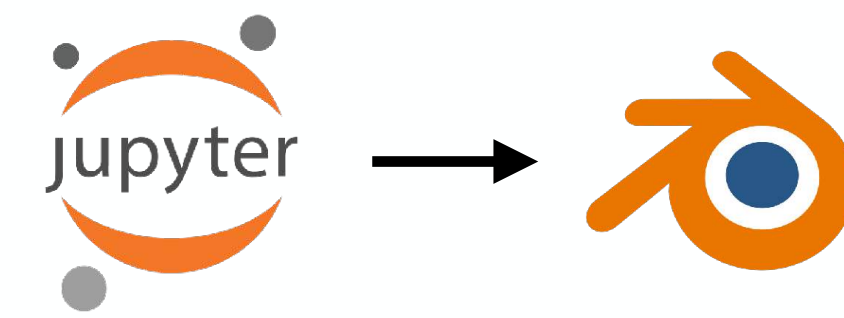
Make Real

100%

```
[15]: import bpy

      # Clear existing objects
      bpy.ops.object.select_all(action='DESELECT')
      bpy.ops.object.select_by_type(type='MESH')
      bpy.ops.object.delete()
```

# GitHub Copilot



**Time Lapse  
Donut with Copilot**

# 2.3. Interactive Tables



```
# Display the DataFrame with the new unique_id column using the quak widget
widget = quak.Widget(df)
widget
```

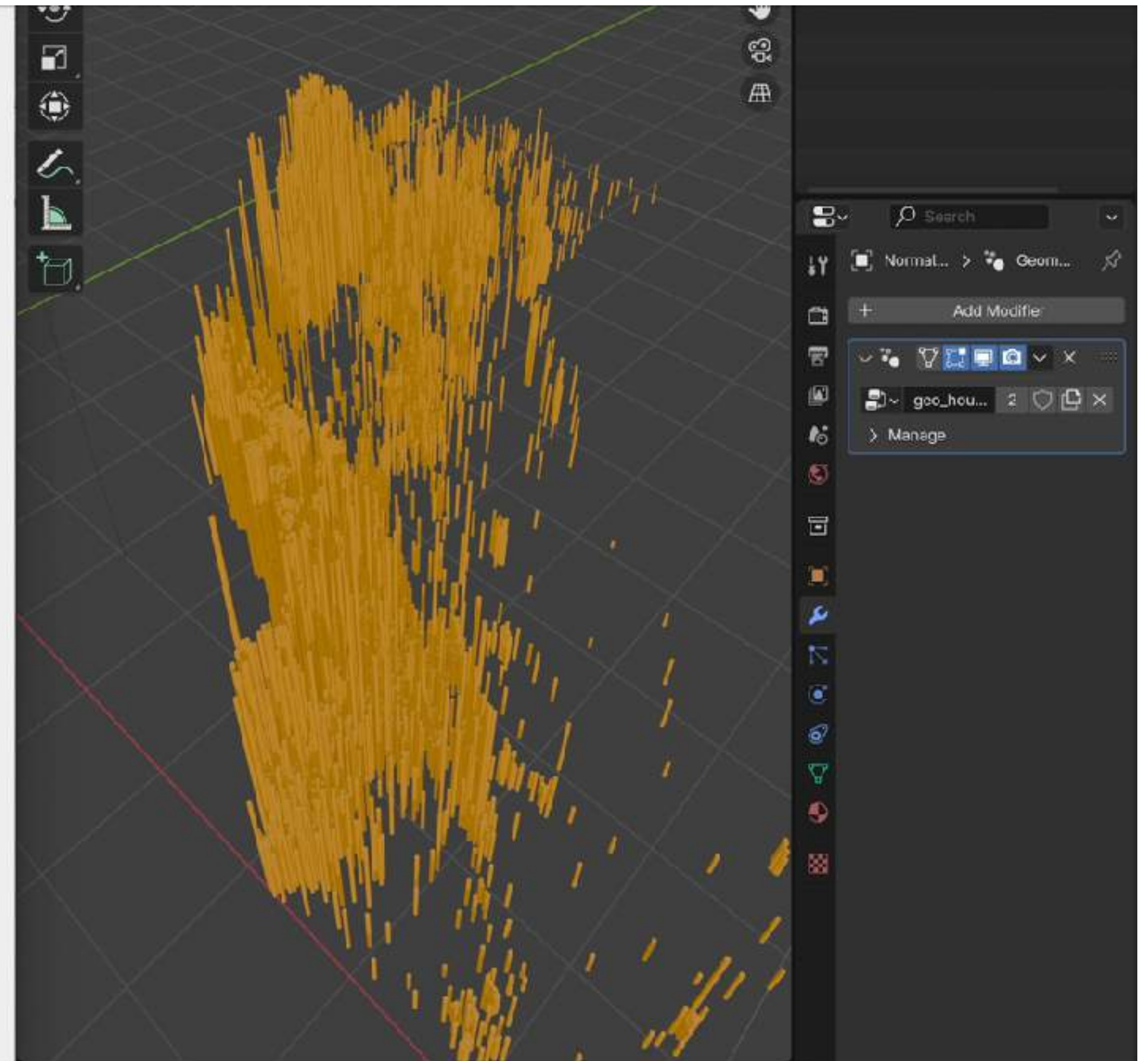
[2]:

total_bedrooms	population	households	median_income	median_hous...	ocean_proxim...
float64	float54	float64	float64	float64	utf8
334	1,119	308	2.217	48,500	INLAND
234	573	236	2.49	74,100	<1H OCEAN
662	1,554	578	2.685	111,300	NEAR OCEAN
1,330	3,418	1,268	4.665	237,800	<1H OCEAN
332	945	292	3.404	115,300	INLAND
1,031	3,221	1,043	7.642	374,900	<1H OCEAN
null	807	218	6.287	325,500	NEAR OCEAN

12,590 rows

[3]:

Log: california\_housing.ipynb X



<https://github.com/manzt/quak>

# 3. Documentation

```
In [6]: fresh_scene()
bpy.ops.mesh.primitive_cylinder_add(radius=1, depth=2, location=(0, 0, 0))
render_result()
```



```
In [7]: fresh_scene()
bpy.ops.mesh.primitive_torus_add(major_radius=2, minor_radius=0.3, location=(0.0, 0))
render_result()
```



```
In [8]: fresh_scene()
bpy.ops.mesh.primitive_ico_sphere_add(radius=1, location=(0, 0, 0))
render_result()
```



```
In [9]: fresh_scene()
bpy.ops.object.text_add(location=(-4, 0, 0))
bpy.context.object.data.body = 'Hello 🍌'
bpy.context.object.scale = (2, 2, 2)
render_result()
```



## Load file

```
In [31]: fresh_scene()
bpy.ops.wm.open_mainfile(filepath='dcrut.blend')
```

Out[31]: {'FINISHED'}

## Choose render engine

```
In [32]: bpy.context.scene.render.engine = 'BLENDER_WORKBENCH'
bpy.context.scene.render.resolution_x = 500
bpy.context.scene.render.resolution_y = 200
render_result()
```



```
In [33]: bpy.context.scene.render.engine = 'BLENDER_EEVEE_NEXT'
render_result()
```



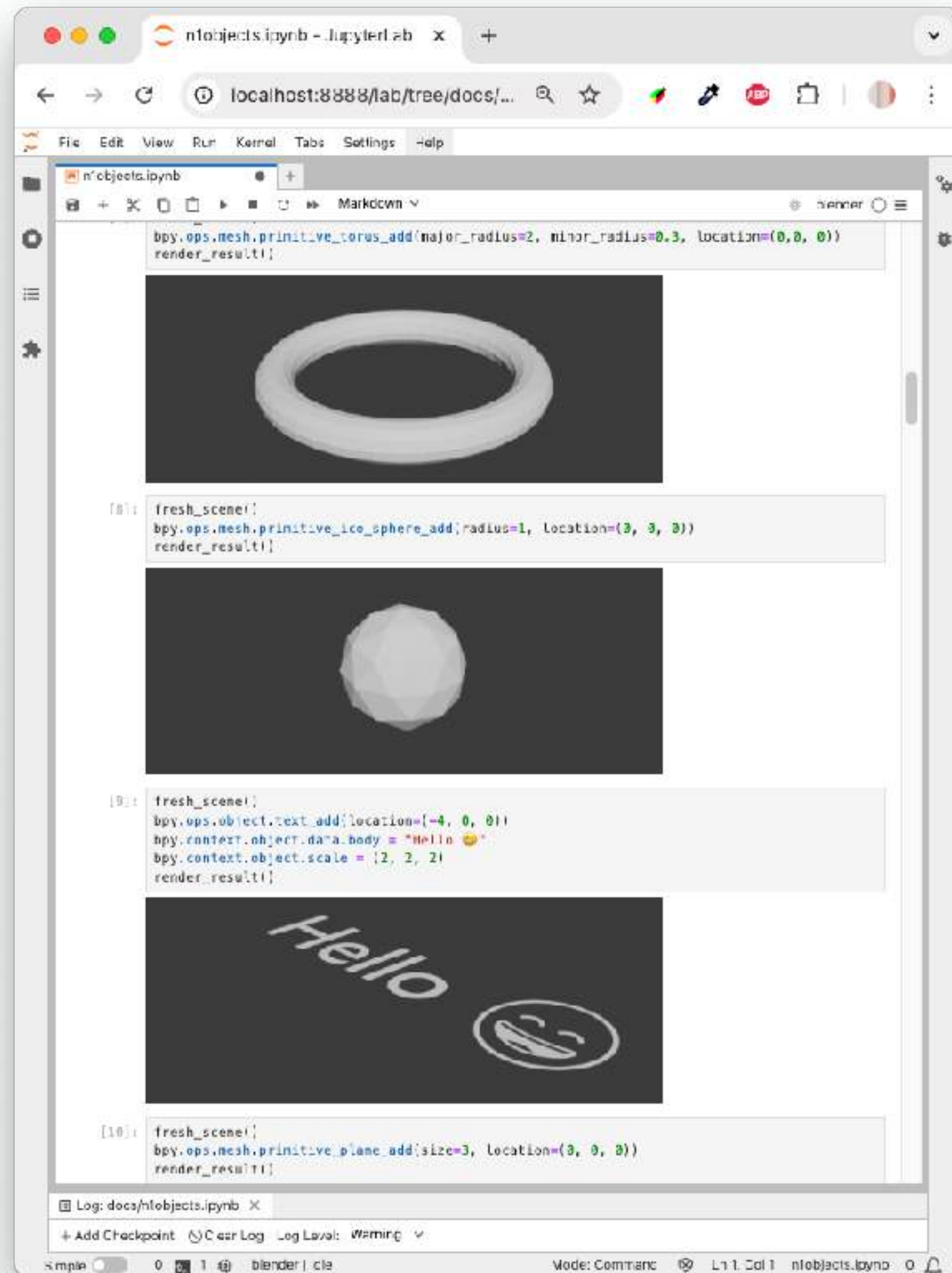
```
In [34]: bpy.context.scene.render.engine = 'CYCLES'
bpy.context.scene.cycles.samples = 100
render_result()
```



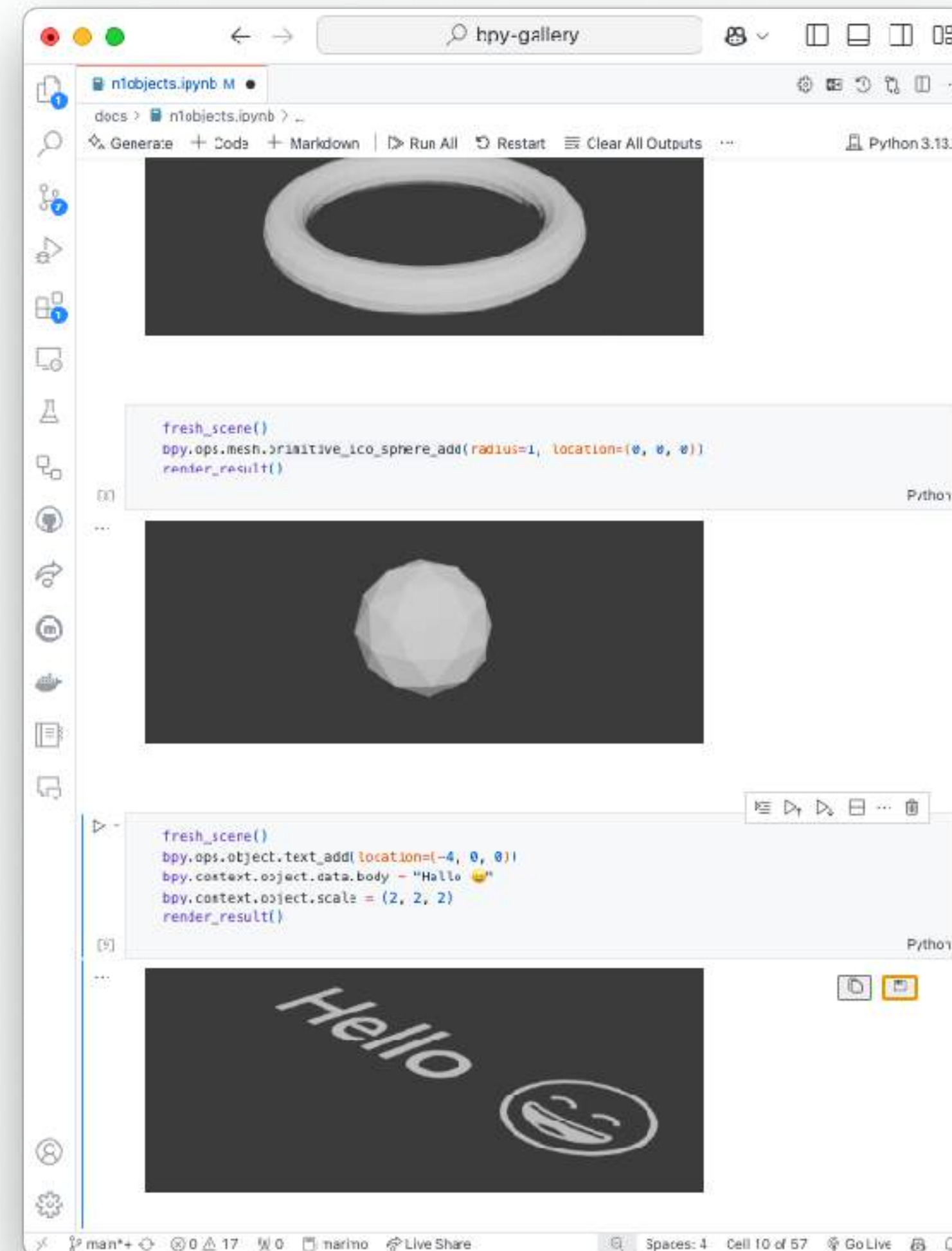
<https://kolibril13.github.io/bpy-gallery/n1objects/>

# Notebook platforms

Jupyter Lab



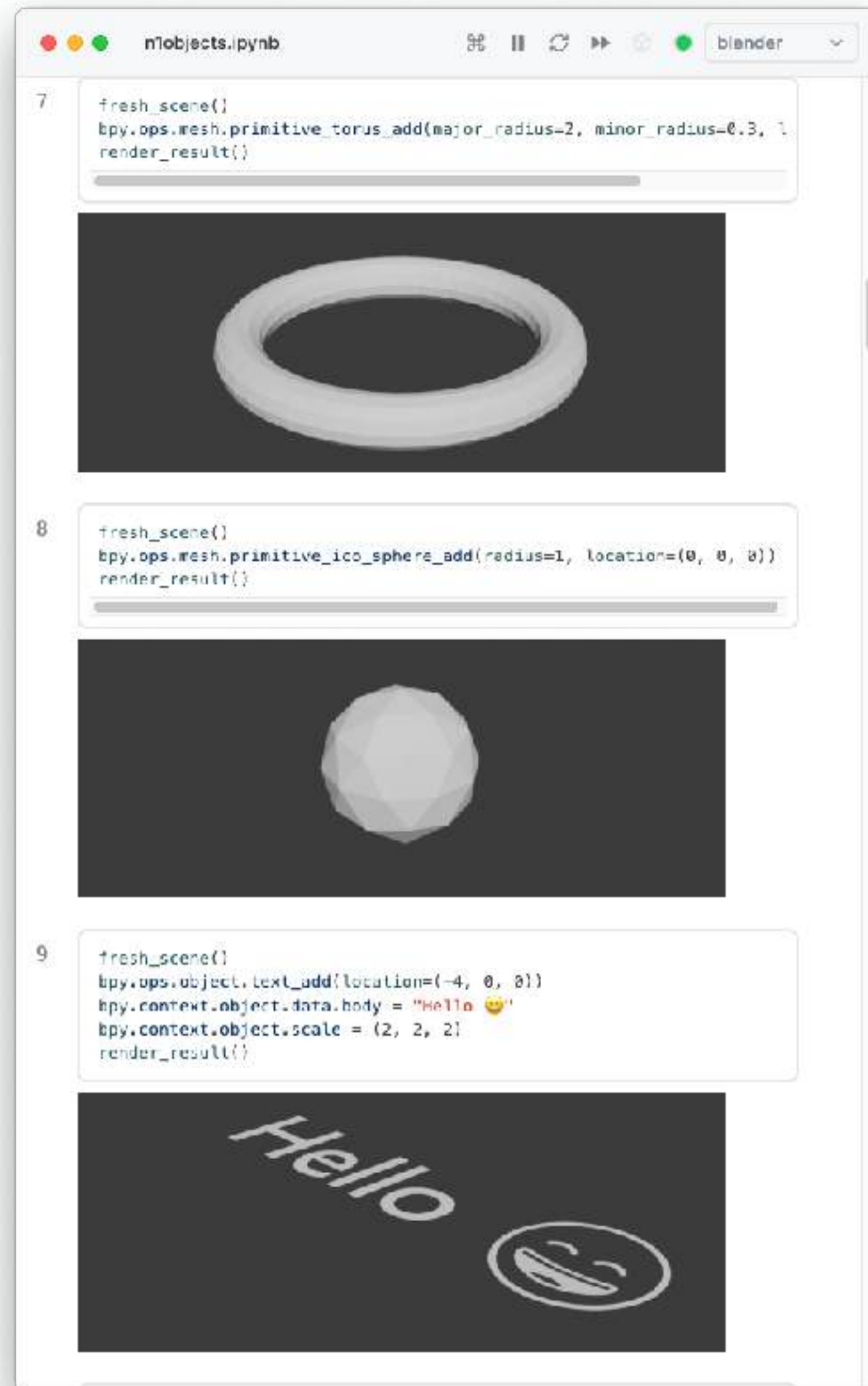
VS Code Notebook





# Notebook platforms

## Satyrn



The Satyrn notebook interface displays three code cells, each followed by its rendered output. The first cell contains code to create a torus, the second to create an icosphere, and the third to create a text object with a smiley face emoji.

```
7 fresh_scene()
bpy.ops.mesh.primitive_torus_add(major_radius=2, minor_radius=0.3, 1)
render_result()
```

```
8 fresh_scene()
bpy.ops.mesh.primitive_ico_sphere_add(radius=1, location=(0, 0, 0))
render_result()
```

```
9 fresh_scene()
bpy.ops.object.text_add(location=(-4, 0, 0))
bpy.context.object.data.body = "Hello 😊"
bpy.context.object.scale = (2, 2, 2)
render_result()
```

## Marimo



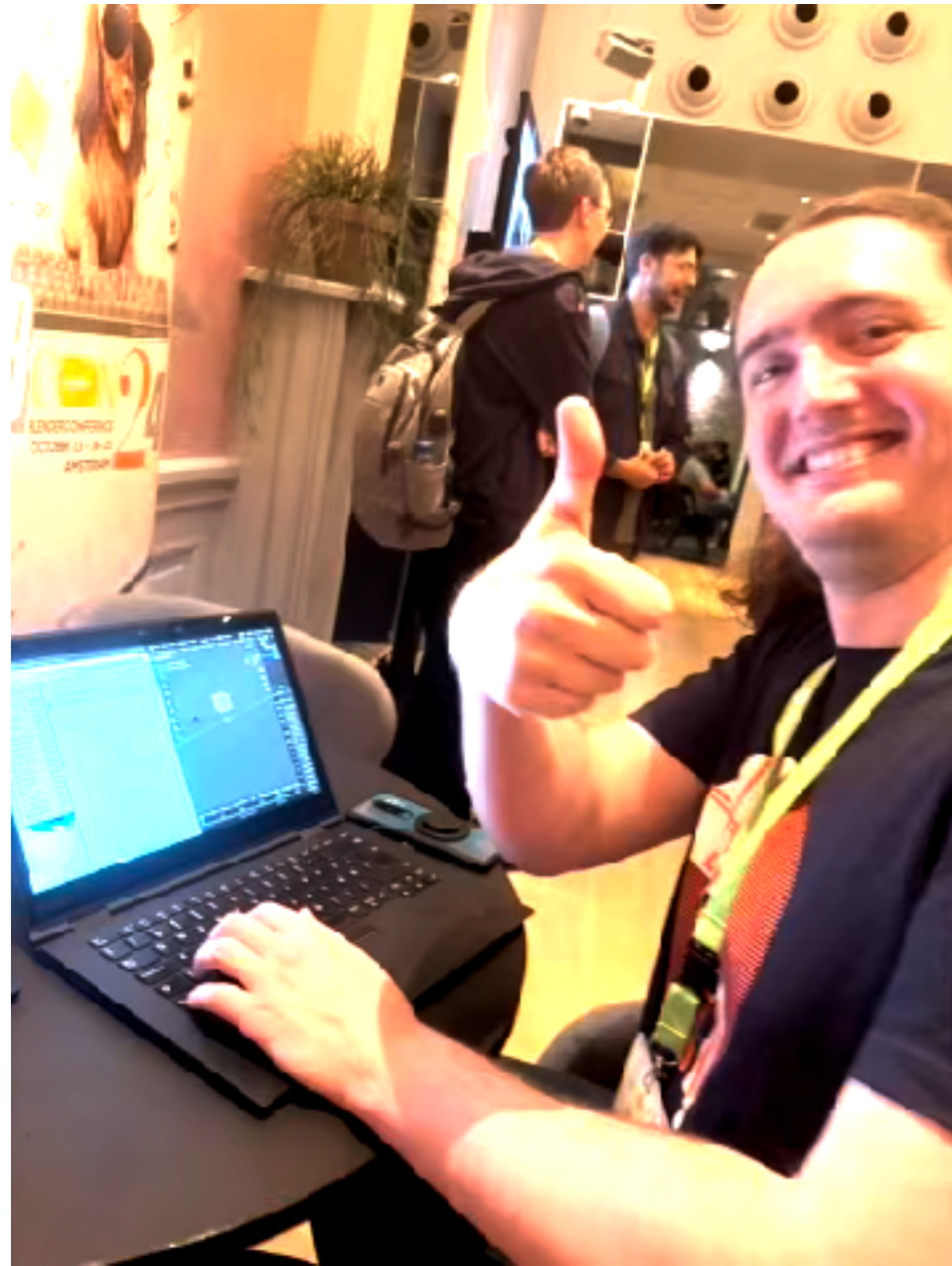
The Marimo notebook interface displays three code cells, each followed by its rendered output. The first cell contains code to create a torus, the second to create an icosphere, and the third to create a text object with a smiley face emoji. The interface includes a browser-like address bar showing localhost:2718 and a sidebar with navigation icons.

```
1 fresh_scene()
2 bpy.ops.mesh.primitive_torus_add(major_radius=2, minor_radius=0.3,
3 location=(0, 0, 0))
render_result()
```

```
1 fresh_scene()
2 bpy.ops.mesh.primitive_ico_sphere_add(radius=1, location=(0, 0, 0))
3 render_result()
```

```
1 fresh_scene()
2 bpy.ops.object.text_add(location=(-4, 0, 0))
3 bpy.context.object.data.body = "Hello 😊"
4 bpy.context.object.scale = (2, 2, 2)
5 render_result()
```

# Installation: bpy\_jupyter Blender Extension



**Sofus Albert Høgsbro Rose**  
so-rose



**Jan-Hendrik Müller**  
kolibril13

[https://github.com/Octoframes/bpy\\_jupyter](https://github.com/Octoframes/bpy_jupyter)

A screenshot of the GitHub repository page for 'bpy\_jupyter' by 'Octoframes'. The page shows the repository name, public status, and various actions like 'Edit Pins', 'Unwatch', 'Fork', and 'Starred'. The main content area displays a list of commits and files, including 'bpy\_jupyter', '.editorconfig', '.gitignore', '.pre-commit-config.ya...', '.python-version', and 'LICENSE'. The right sidebar contains an 'About' section with details like 'Blender extension exposing a jupyter kernel within Blender.', 'Readme', 'AGPL-3.0 license', 'Activity', 'Custom properties', '3 stars', '2 watching', '0 forks', and 'Report repository'.

# Installation

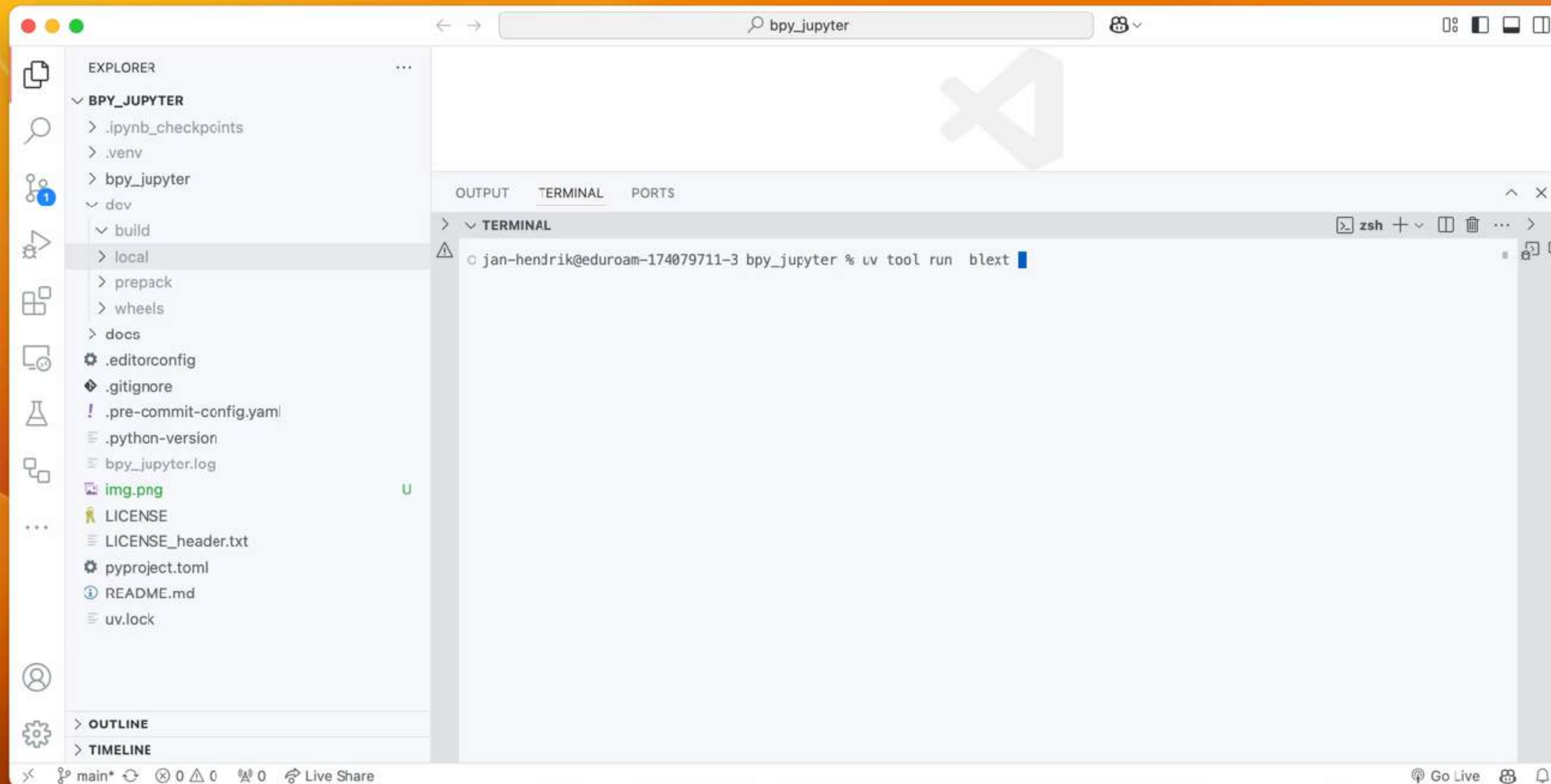
Soon on:

<https://extensions.blender.org/>

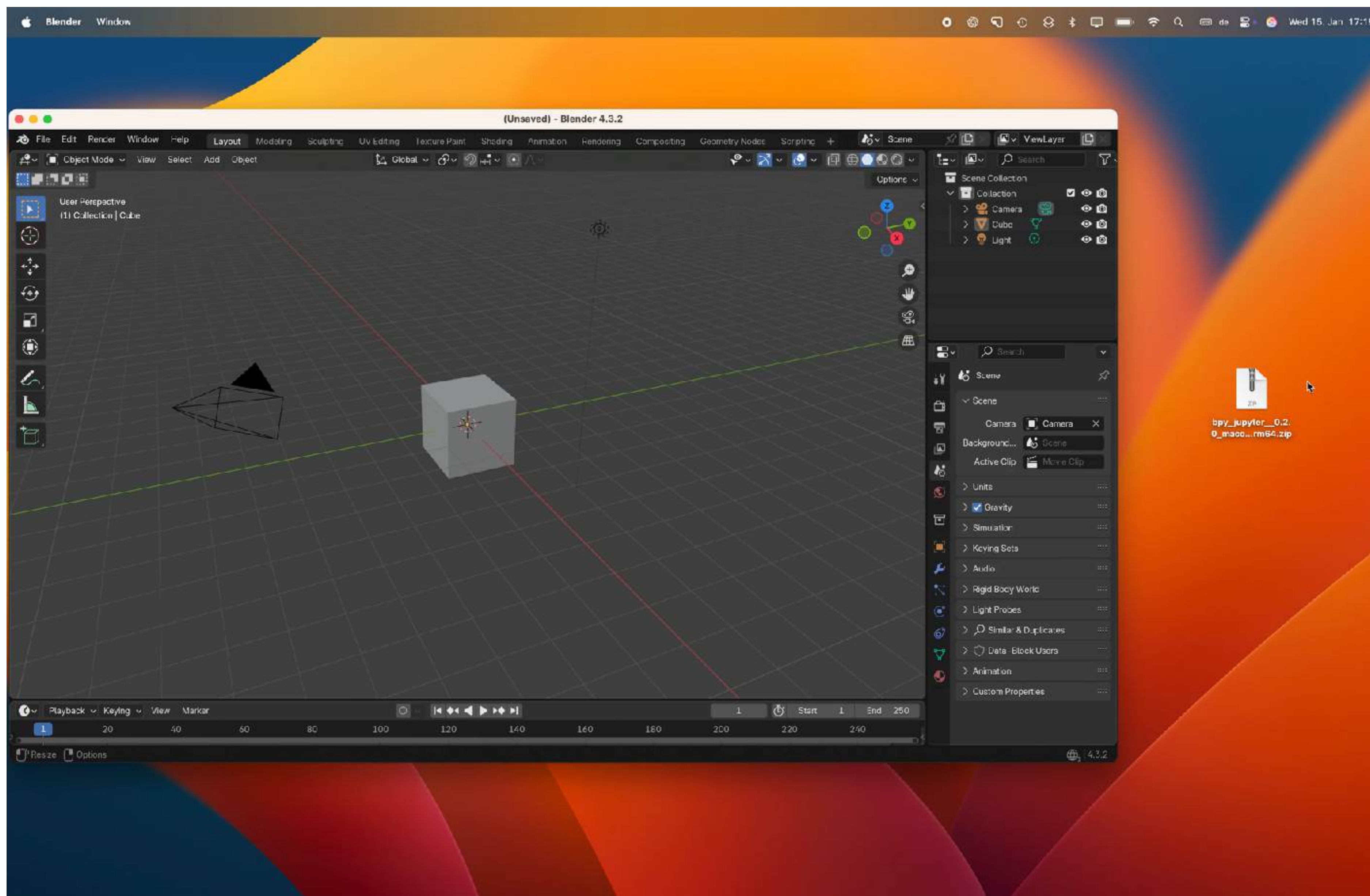
Currently:

1. Clone [https://github.com/Octoframes/bpy\\_jupyter](https://github.com/Octoframes/bpy_jupyter)
2. Install via **uv tool run blext dev**

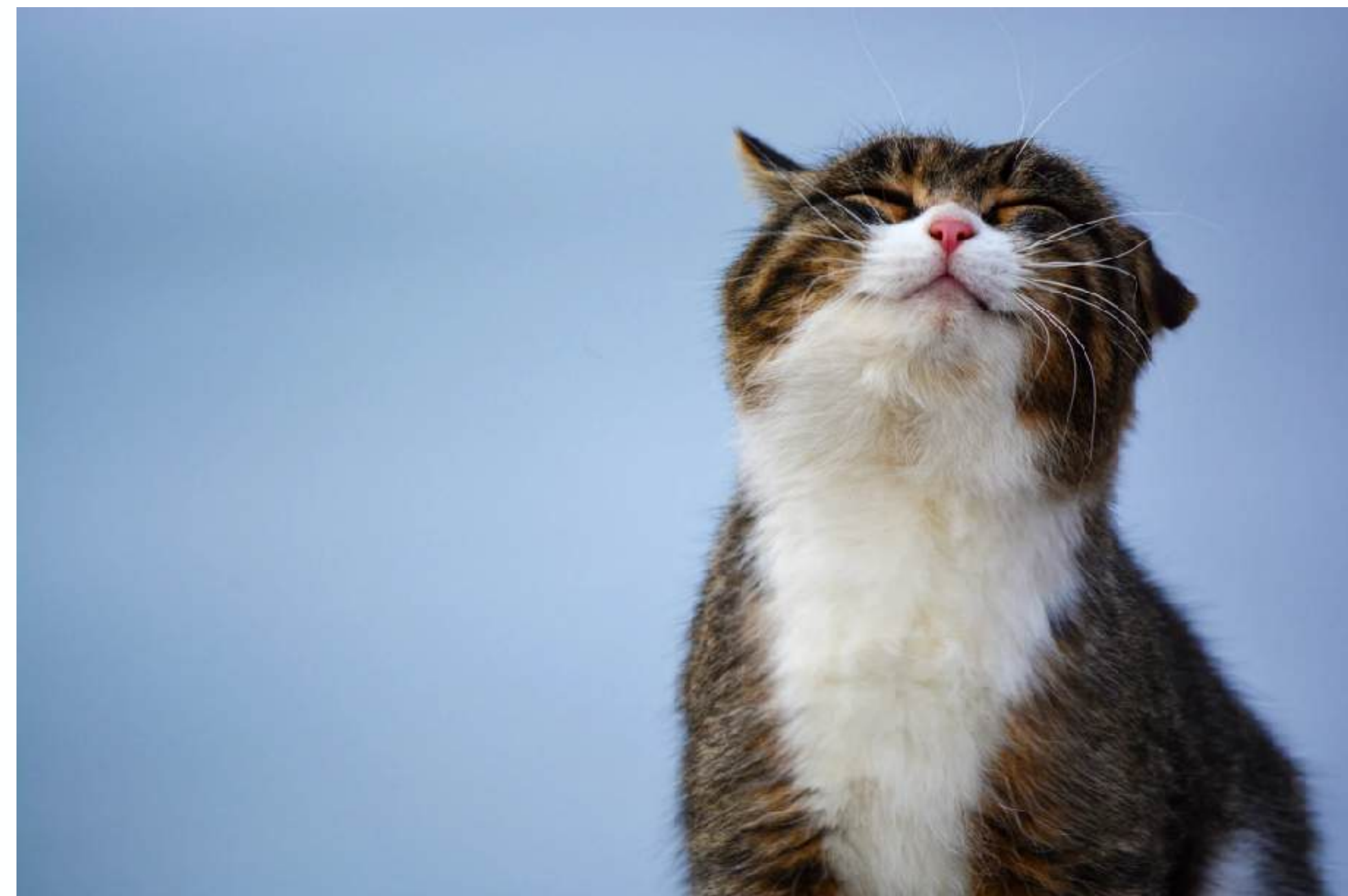
# Build extension



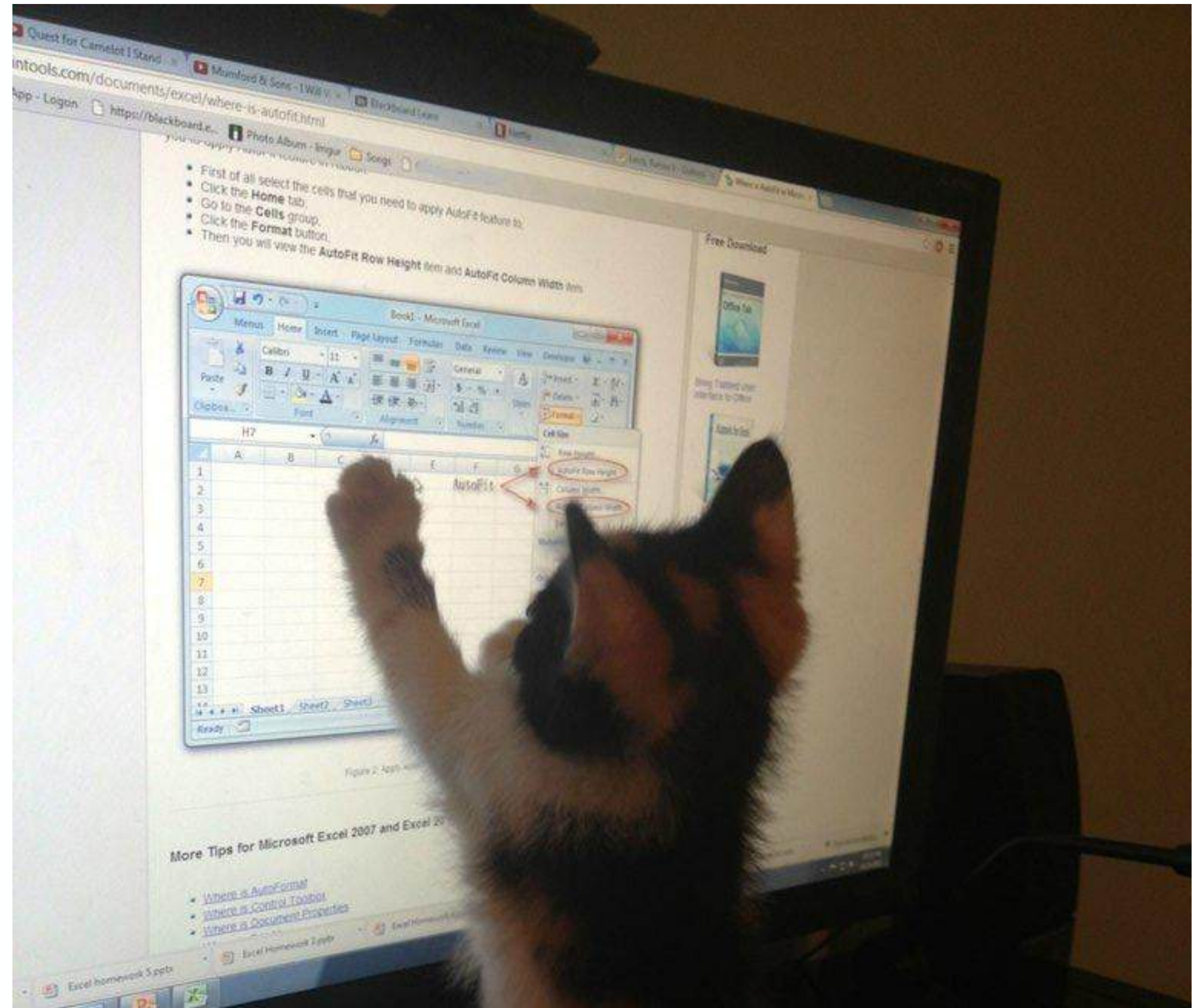
# Install+Connect to JupyterLab



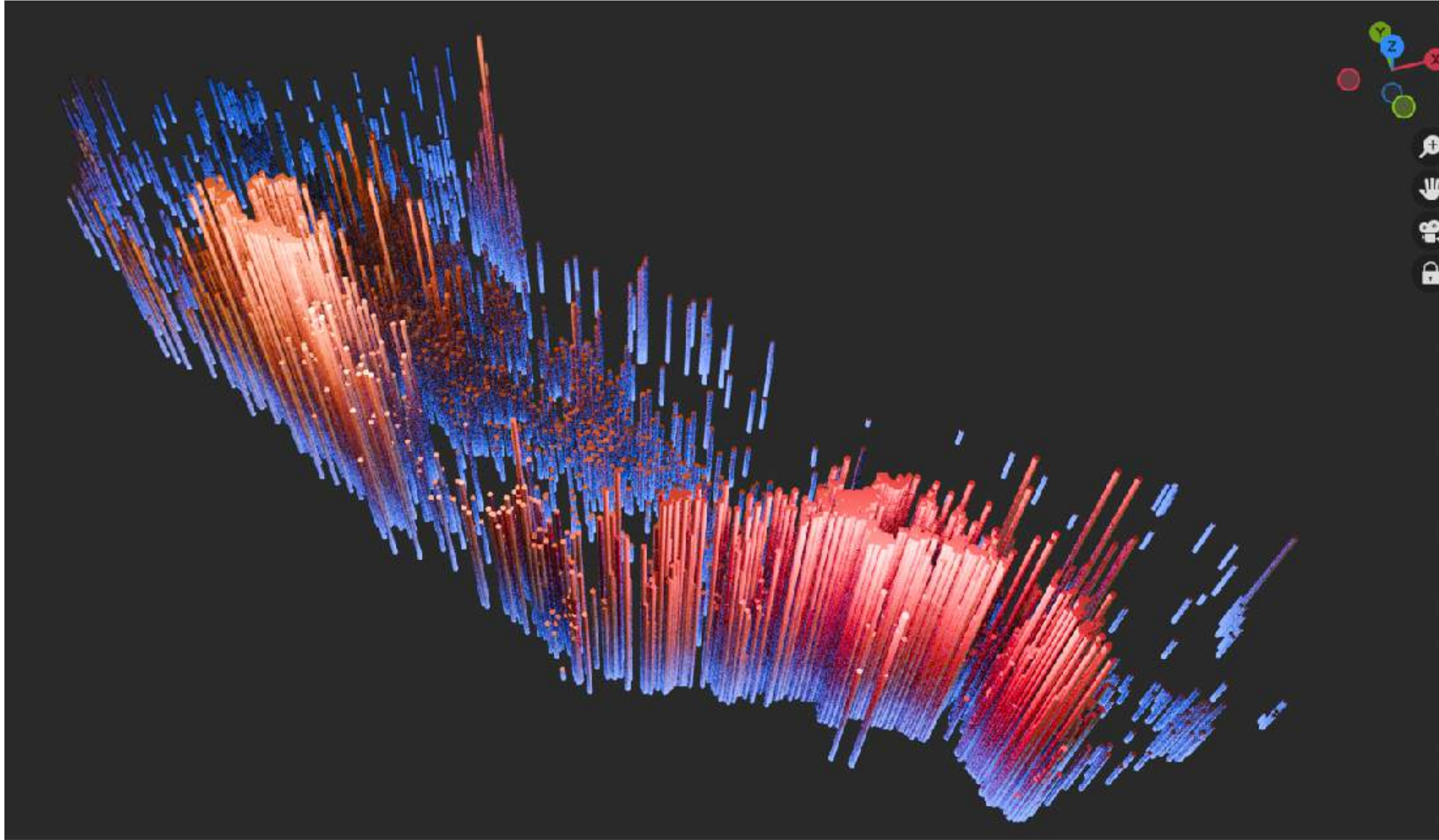
**Demo time!**



# What about spreadsheets?



# Data Processing Pipeline in Blender



California dataset: Latitude, longitude and house prices (12k datapoints)

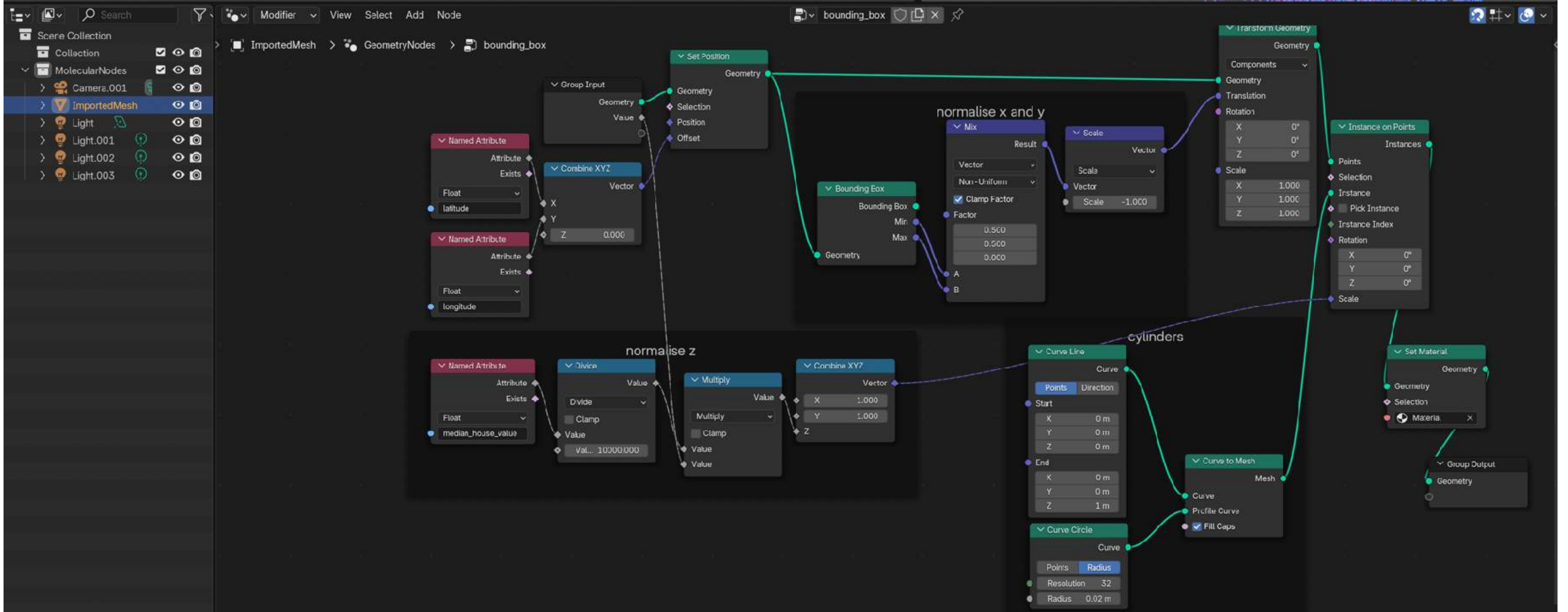


File Edit Render Window Help    Layout Shading **Geometry Nodes** Scripting +    Scene    ViewLayer

Original ImportedMesh

	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	20.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.691	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1294.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2669.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	769.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	514000.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	924000.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	1036000.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	1897000.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	3994000.000

Rows: 12,590 | Columns: 10

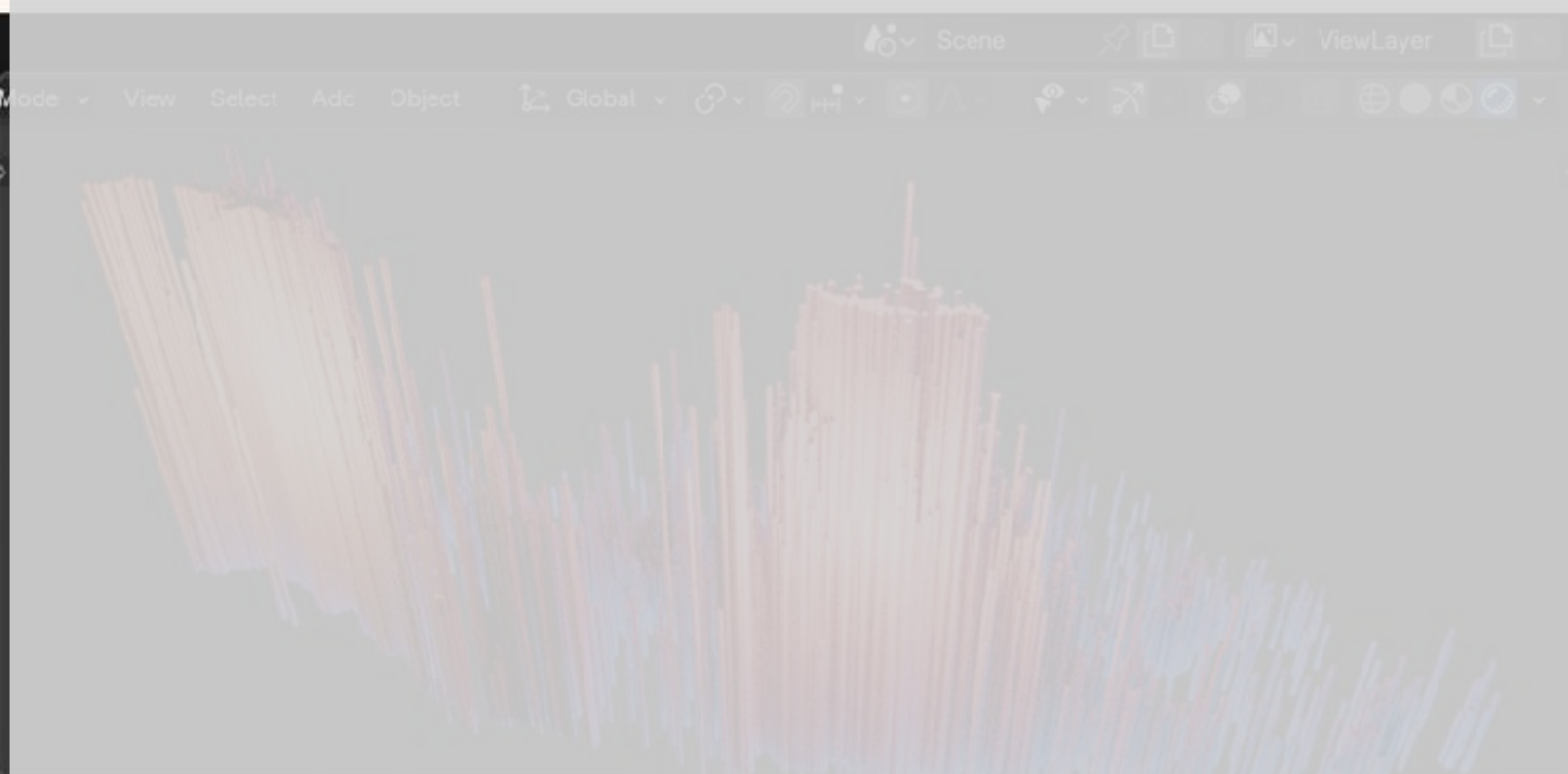


File Edit Render Window Help Layout Shading Geometry Nodes Scripting +

Original ImportedMesh

	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	20.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.691	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1294.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2669.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	769.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	51400.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10



# Data with Python

```
[1]: import numpy as np
import databpy as db

vertex = np.array([[0, 0, 0]])
obj = db.create_bob(vertex, name="Mesh1")
obj.name
```

```
[1]: 'Mesh1'
```

<https://github.com/bradyajohnston/databpy>

README GPL-3.0 license

## databpy

codecov 61% pypi v0.0.8 Run Tests passing ci-cd.yml passing

A set of data-oriented wrappers around the python API of Blender.



**Brady Johnston**  
BradyAJohnston · he/him

# Creating Meshes

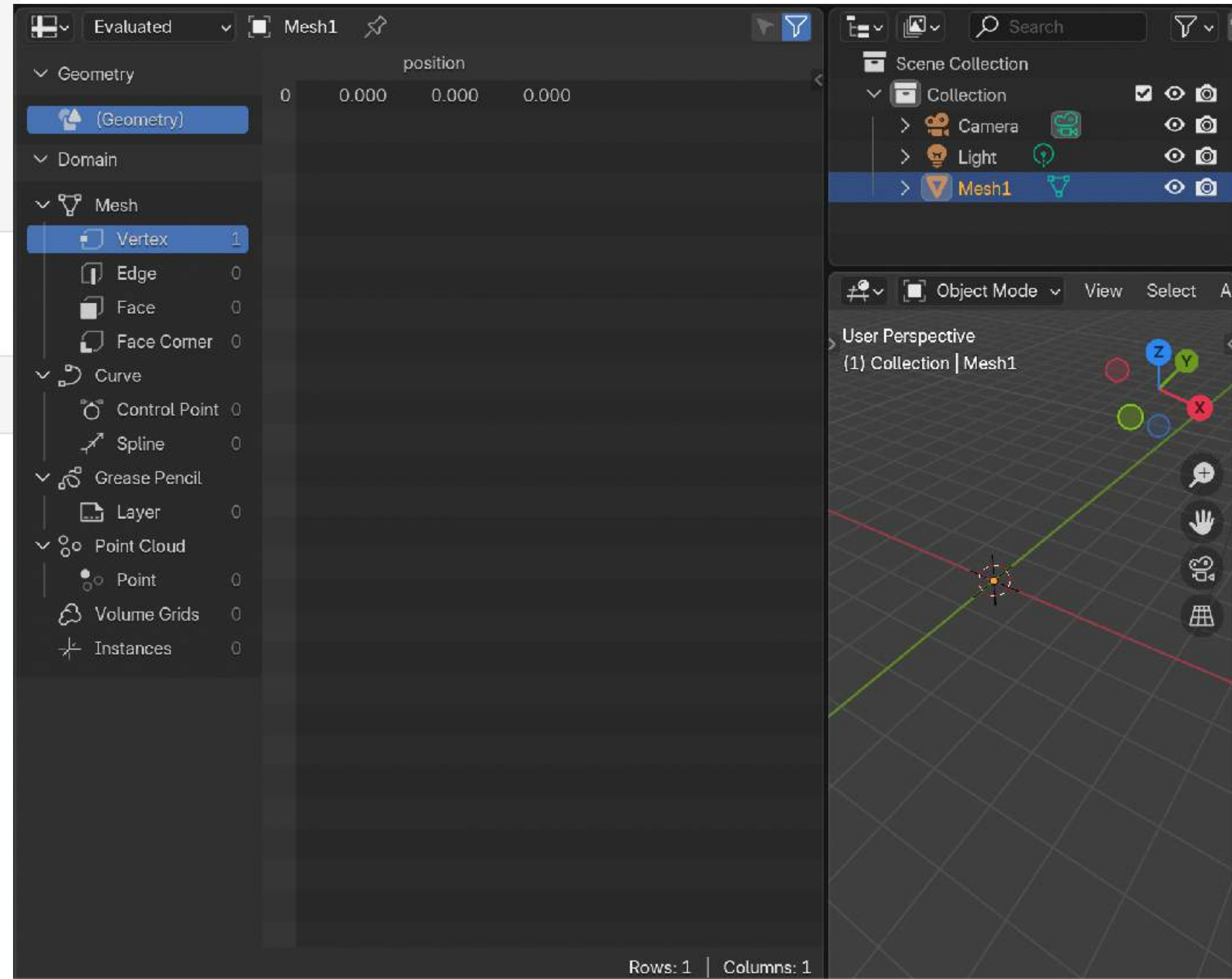
```
import numpy as np
import databpy as db
```

```
vertex = np.array([[0, 0, 0]])
obj = db.create_bob(vertex, name="Mesh1")
obj.name
```

'Mesh1'

```
obj.position
```

```
array([[0., 0., 0.]])
```



# Creating Meshes

```
import numpy as np
import databpy as db
```

```
coords = np.random.rand(20, 3)
obj = db.create_bob(coords, name="Mesh2")
obj.name
```

'Mesh2'

```
obj.position
```

```
array([[0.41664836, 0.35038018, 0.63478756],
       [0.11875096, 0.31862018, 0.12424635],
       [0.28274721, 0.43855155, 0.6550467 ],
       [0.80807185, 0.01969962, 0.99721068],
       [0.33697239, 0.27726692, 0.7743066 ],
       [0.2434269 , 0.29881901, 0.34785211],
       [0.47011954, 0.91141248, 0.20741948],
       [0.96200317, 0.31830707, 0.29814231],
       [0.66150445, 0.88354313, 0.18802765],
       [0.79731137, 0.51676905, 0.46561515],
       [0.60903811, 0.51614606, 0.50760353],
```

The screenshot shows a 3D software interface with a dark theme. On the left, a sidebar contains a hierarchy tree with categories like Geometry, Domain, Mesh, Curve, Grease Pencil, Point Cloud, and Volume Grids. The 'Mesh' category is expanded, showing 'Vertex' with a count of 20. In the center, a table displays the 'position' data for each vertex, with columns for index and three coordinates. On the right, another sidebar shows the 'Scene Collection' with objects like Camera, Light, Mesh1, and Mesh2. Below that, the 'Object Mode' is active, showing a 3D view of the mesh as a collection of orange points on a grid. The bottom right corner indicates 'Rows: 20 | Columns: 1'.

	position
0	0.417 0.350 0.635
1	0.119 0.319 0.124
2	0.283 0.439 0.655
3	0.808 0.020 0.997
4	0.337 0.277 0.774
5	0.243 0.299 0.348
6	0.470 0.911 0.207
7	0.962 0.318 0.298
8	0.662 0.884 0.188
9	0.797 0.517 0.466
10	0.609 0.516 0.508
11	0.680 0.992 0.484
12	0.281 0.108 0.599
13	0.582 0.684 0.850
14	0.131 0.160 0.537
15	0.964 0.225 0.604
16	0.171 0.764 0.475
17	0.703 0.142 0.289
18	0.283 0.021 0.869
19	0.490 0.560 0.183

# Creating Meshes

```
%%time
import numpy as np
import databpy as db

# Generate 2 million random 3D points
coords = np.random.rand(2_000_000, 3)

# Create a mesh object using databpy
obj = db.create_bob(coords, name="Mesh2M")
print(obj.name)

Mesh2M
CPU times: user 685 ms, sys: 36.3 ms, total:
Wall time: 717 ms
```

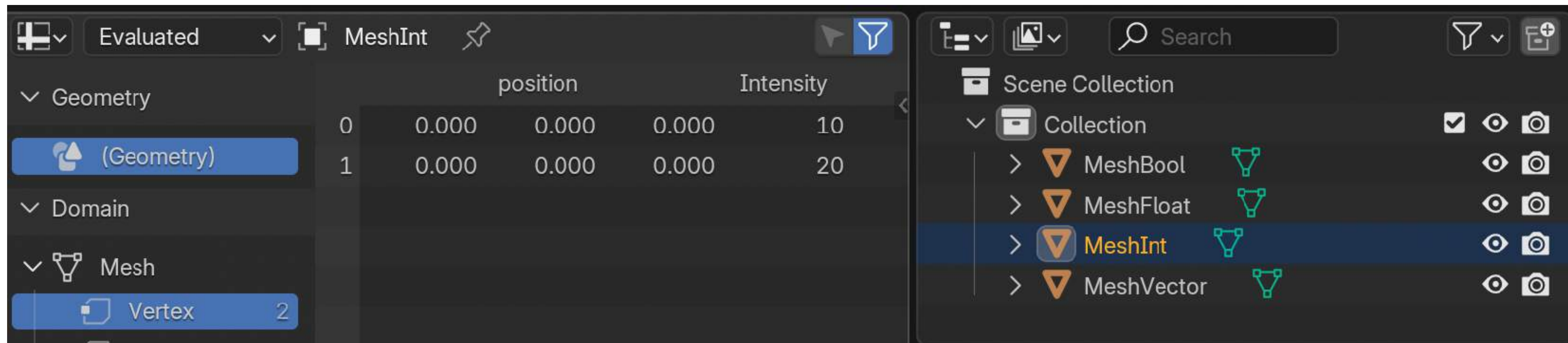
The image shows a Blender 2.80 interface with a 3D scene. The main viewport displays a large, orange, rectangular mesh object named 'Mesh2M'. The interface includes a top status bar with 'Evaluated' and 'Mesh2M', a left sidebar with the Outliner and Properties panels, and a right sidebar with the Outliner and Properties panels. The 3D viewport shows the mesh in a perspective view with a coordinate system and a navigation toolbar. The Outliner on the right shows the scene collection and the 'Mesh2M' object selected. The Properties panel on the right shows the 'Mesh' properties for the selected object. The Outliner on the left shows the hierarchy of the scene, including 'Geometry', 'Domain', 'Mesh', 'Curve', 'Grease Pencil', 'Point Cloud', and 'Instances'. The 'Mesh' panel shows the 'Vertex' count as 2.0M. The 3D viewport shows the mesh in a perspective view with a coordinate system and a navigation toolbar. The status bar at the bottom indicates 'Rows: 2,000,000 | Columns: 1'.

	position
0	0.048 0.047 0.288
1	0.649 0.559 0.112
2	0.104 0.098 0.030
3	0.752 0.473 0.933
4	0.580 0.932 0.044
5	0.550 0.813 0.750
6	0.455 0.436 0.886
7	0.100 0.180 0.849
8	0.413 0.889 0.027
9	0.401 0.219 0.982
10	0.339 0.170 0.770
11	0.239 0.047 0.241
12	0.377 0.038 0.917
13	0.125 0.088 0.488
14	0.390 0.392 0.229
15	0.932 0.519 0.068
16	0.955 0.521 0.697
17	0.048 0.107 0.432
18	0.522 0.611 0.684
19	0.798 0.590 0.690
20	0.387 0.158 0.766
21	0.925 0.644 0.482
22	0.728 0.966 0.604
23	0.983 0.787 0.743
24	0.880 0.453 0.843
25	0.392 0.777 0.721
26	0.639 0.267 0.096
27	0.487 0.423 0.210

# Attributes to each point

```
import numpy as np
import polars as pl
from csv_importer.parsers import polars_df_to_bob

df = pl.DataFrame(
    {
        "Intensity": [10, 20],
    }
)
bob = polars_df_to_bob(df, name="MeshInt")
```



The screenshot shows a software interface with two main panels. The left panel displays a data table with columns for 'position' and 'Intensity'. The right panel shows a 'Scene Collection' with a list of mesh types: MeshBool, MeshFloat, MeshInt, and MeshVector. The 'MeshInt' entry is highlighted.

	position	Intensity
0	0.000 0.000 0.000	10
1	0.000 0.000 0.000	20

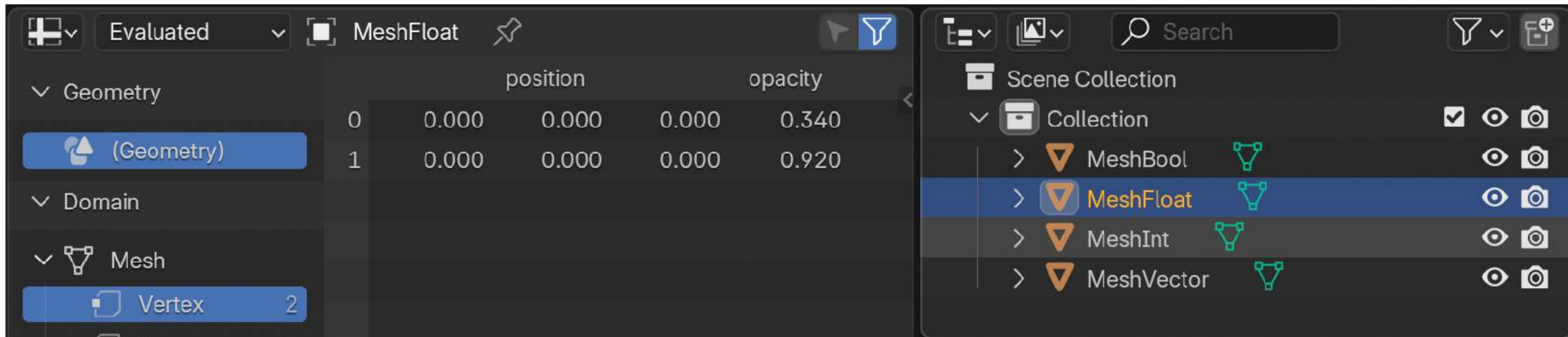
Scene Collection

- Collection
  - MeshBool
  - MeshFloat
  - MeshInt**
  - MeshVector

# Attributes to each point

```
import numpy as np
import polars as pl
from csv_importer.parsers import polars_df_to_bob

df = pl.DataFrame(
    {
        "opacity": [0.34, 0.92],
    }
)
bob = polars_df_to_bob(df, name="MeshFloat")
```



The image shows a Blender interface with two panels. The left panel displays the 'MeshFloat' object in the Outliner, showing its geometry and domain. The right panel shows the 'Scene Collection' containing a 'Collection' with four mesh objects: MeshBool, MeshFloat, MeshInt, and MeshVector.

Index	position	opacity
0	0.000 0.000 0.000	0.340
1	0.000 0.000 0.000	0.920

Scene Collection

- Collection
  - MeshBool
  - MeshFloat**
  - MeshInt
  - MeshVector



# Attributes to each point

```
import numpy as np
import polars as pl
from csv_importer.parsers import polars_df_to_bob

df = pl.DataFrame(
    {
        "Is_Visible": [True, False],
    }
)
bob = polars_df_to_bob(df, name="MeshBool")
```

The screenshot displays a software interface with two main panels. The left panel shows a data table with columns for 'position' and 'Is\_Visible'. The right panel shows a scene collection with a 'MeshBool' object highlighted.

	position	Is_Visible
0	0.000 0.000 0.000	<input checked="" type="checkbox"/>
1	0.000 0.000 0.000	<input type="checkbox"/>

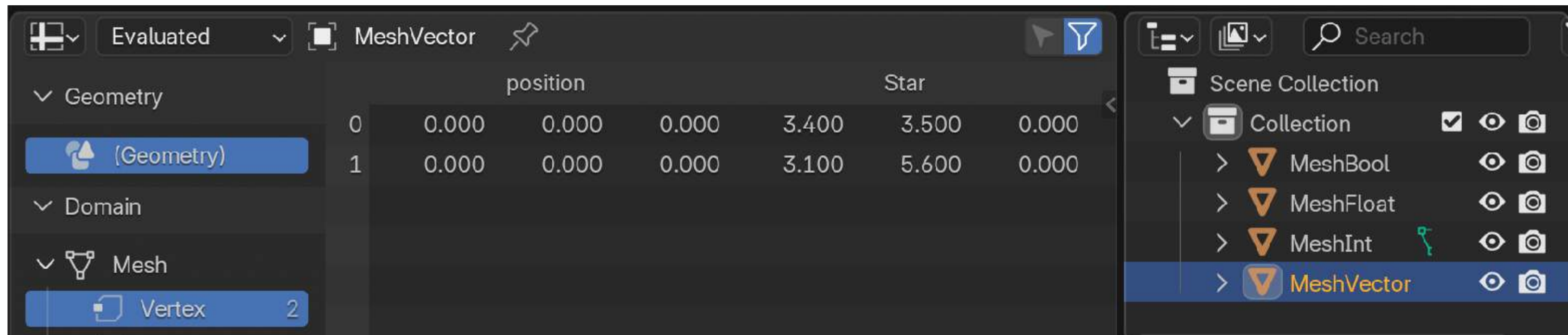
The scene collection on the right includes:

- Scene Collection
- Collection
  - MeshBool (highlighted)
  - MeshFloat
  - MeshInt
  - MeshVector

# Attributes to each point

```
import numpy as np
import polars as pl
from csv_importer.parsers import polars_df_to_bob

df = pl.DataFrame({
    "Star": [
        [3.4, 3.5, 0.0],
        [3.1, 5.6, 0.0]
    ]
})
obj = polars_df_to_bob(df, name="MeshVector")
```



The screenshot shows a software interface with a table of data for a 'MeshVector' object. The table has columns for 'position' and 'Star'. The 'Star' column contains two rows of data: [3.4, 3.5, 0.0] and [3.1, 5.6, 0.0]. The interface also shows a sidebar with a 'Scene Collection' and a 'Collection' containing 'MeshVector'.

	position	Star
0	0.000 0.000 0.000	3.400 3.500 0.000
1	0.000 0.000 0.000	3.100 5.600 0.000

# Attributes to each point

```
import numpy as np
import polars as pl
from csv_importer.parsers import polars_df_to_bob

df = pl.DataFrame({
    "String": ["Hello", "World"],
})
# Stings are not supported
obj = polars_df_to_bob(df, name="MeshVector")
```

Automatically skipped





# Attributes to each point

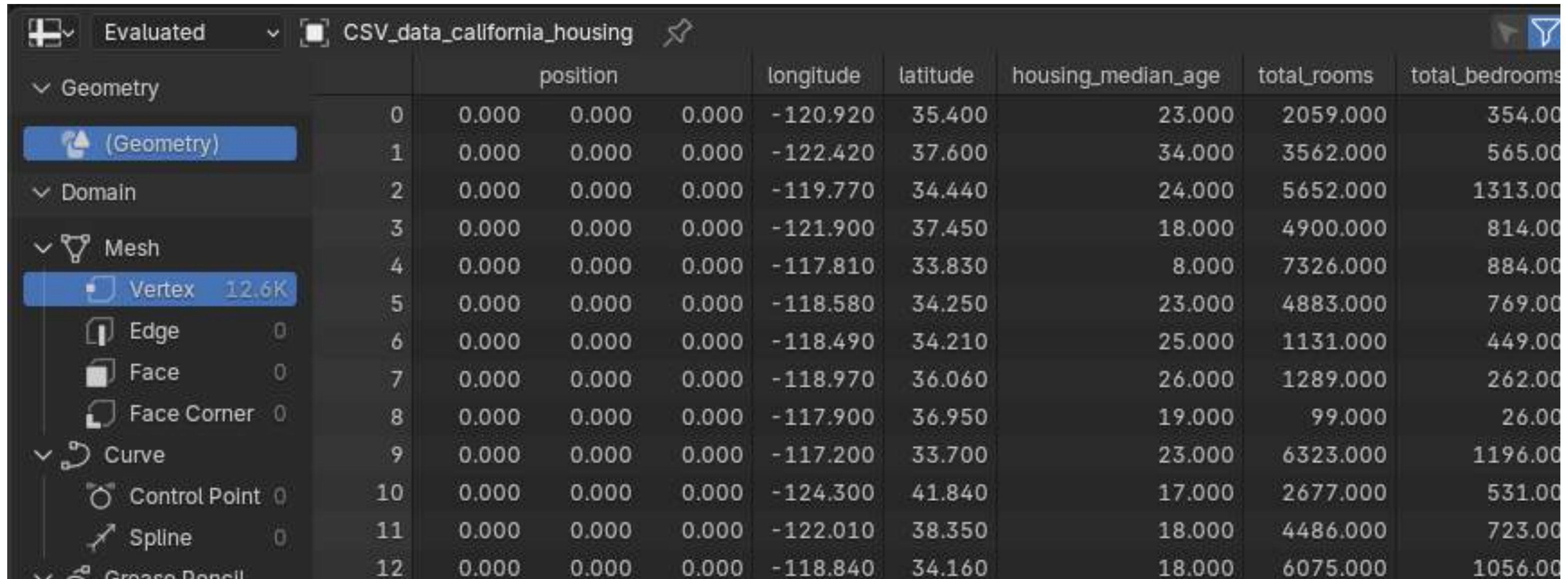
```
: from pathlib import Path
from csv_importer.csv import load_csv

csv_path = Path.home() / "projects" / "blender_csv_import" / "docs" / "sample_datasets" / "data_california_housing.csv"

# `load_csv` is like `like polars_df_to_bob`, but with a csv path input
obj = load_csv(csv_path)

print(obj.name)
```

CSV\_data\_california\_housing



	position	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	0.000 0.000 0.000	-120.920	35.400	23.000	2059.000	354.000
1	0.000 0.000 0.000	-122.420	37.600	34.000	3562.000	565.000
2	0.000 0.000 0.000	-119.770	34.440	24.000	5652.000	1313.000
3	0.000 0.000 0.000	-121.900	37.450	18.000	4900.000	814.000
4	0.000 0.000 0.000	-117.810	33.830	8.000	7326.000	884.000
5	0.000 0.000 0.000	-118.580	34.250	23.000	4883.000	769.000
6	0.000 0.000 0.000	-118.490	34.210	25.000	1131.000	449.000
7	0.000 0.000 0.000	-118.970	36.060	26.000	1289.000	262.000
8	0.000 0.000 0.000	-117.900	36.950	19.000	99.000	26.000
9	0.000 0.000 0.000	-117.200	33.700	23.000	6323.000	1196.000
10	0.000 0.000 0.000	-124.300	41.840	17.000	2677.000	531.000
11	0.000 0.000 0.000	-122.010	38.350	18.000	4486.000	723.000
12	0.000 0.000 0.000	-118.840	34.160	18.000	6075.000	1056.000

# Also GUI version for CSV data

The screenshot shows the Blender Extensions website for the CSV Importer add-on. The page features a dark theme and includes a navigation bar with links for Extensions, Add-ons, Themes, Approval Queue, and About. A search bar and an 'Upload Extension' button are also present. The main content area displays the add-on's name, 'CSV Importer', and a brief description: 'import csv data to meshes in Blender. Add-on by Jan-Hendrik-Müller'. Below this, there are tabs for 'About', 'What's New', 'Permissions', 'Reviews', and 'Version History'. A large image shows a 3D visualization of data as a mesh of colored lines. To the right, a sidebar provides details about the developer (Jan-Hendrik-Müller), a 4-star rating, version 0.1.6, and a publication date of Dec. 9th, 2024. It also shows 2738 downloads and compatibility with Blender 4.2.5 and newer. The supported platforms listed are macOS (Apple Silicon, Intel), Windows, and Linux. A link to the developer's GitHub repository is provided.

extensions.blender.org/add-ons/csv-importer/

Extensions Add-ons Themes Approval Queue About

Search... Upload Extension

< All Add-ons

**CSV Importer**  
import csv data to meshes in Blender.  
Add-on by Jan-Hendrik-Müller

About What's New Permissions Reviews Version History Edit

Developer  
[Jan-Hendrik-Müller](#)

Rating  
★★★★★ (4)

Version  
[0.1.6](#)

Updated  
1 d

Published  
Dec. 9th, 2024

Downloads  
2738

Compatibility  
[Blender 4.2.5](#) and newer

Supported Platforms  
macOS Apple Silicon, Intel  
Windows  
Linux

Website  
[github.com/kolibri13/blender\\_csv\\_import](https://github.com/kolibri13/blender_csv_import)

Report Issues

	longitude	latitude	housing_median_age	population
4	-117.810	33.830	8.000	2569.000
5	-118.580	34.250	23.000	2119.000
6	-118.490	34.210	25.000	746.000
7	-118.970	36.060	26.000	1100.000
8	-117.900	36.950	19.000	51.000

<https://extensions.blender.org/add-ons/csv-importer/>

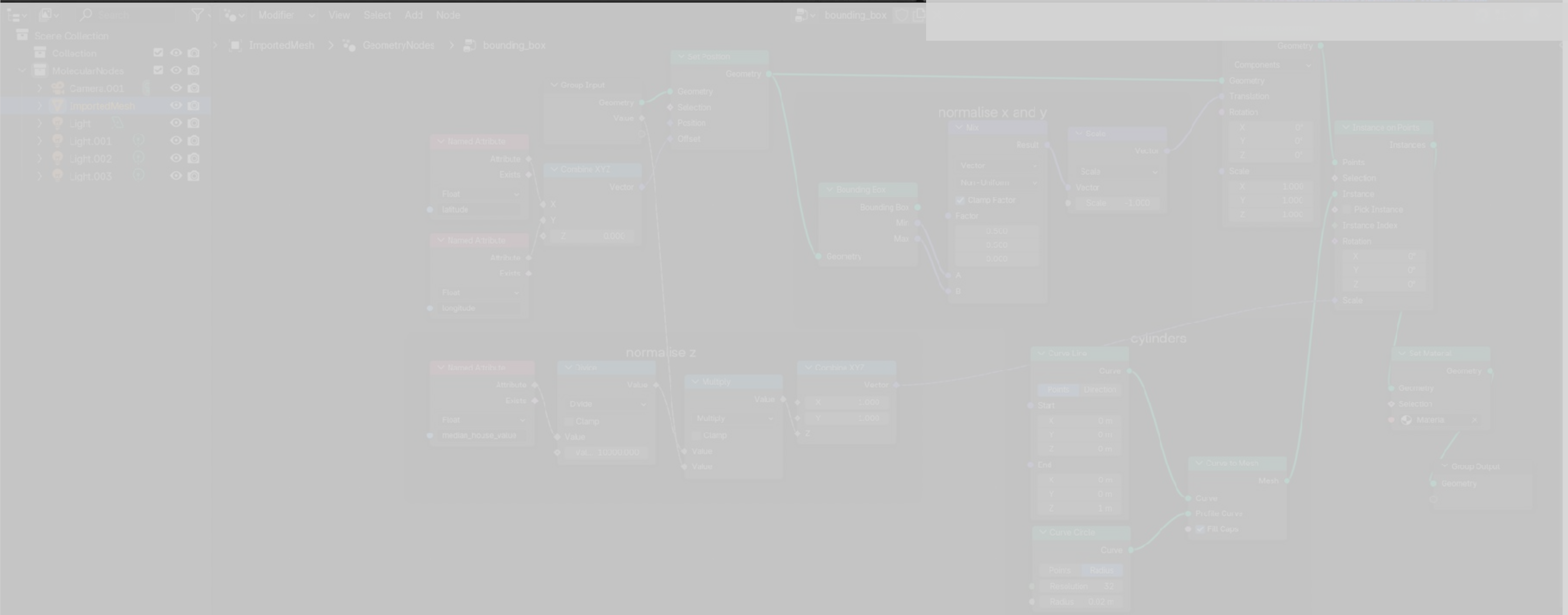
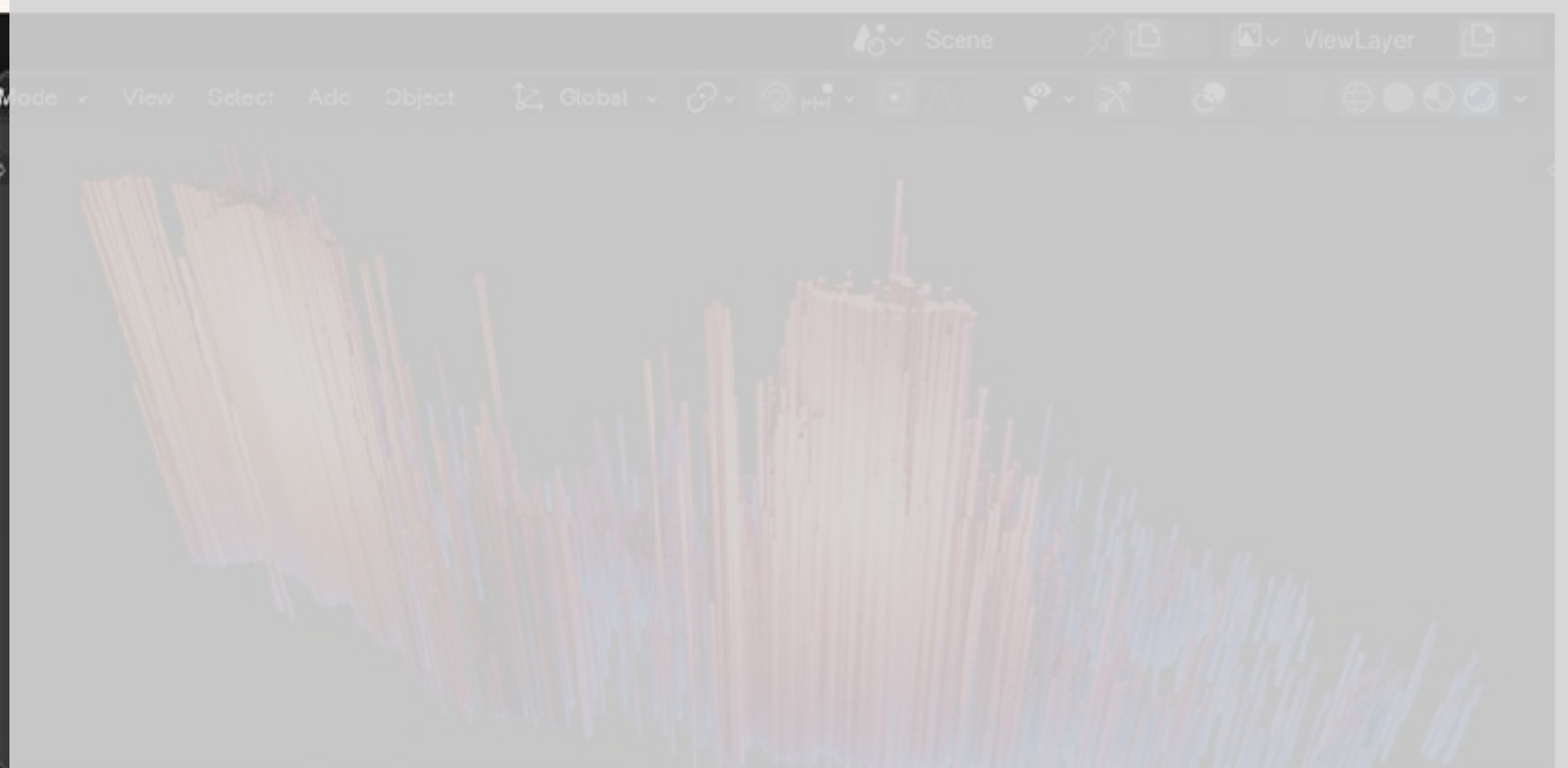
File Edit Render Window Help Layout Shading Geometry Nodes Scripting +

Original ImportedMesh

	gitude	latitude	housing_median_age	total_rocms	total_bedrooms	population	househo.ds	median_income	median_house_value
0	20.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.691	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1294.000	2.472	295300.000
3	21.900	37.450	18.000	732.000	88.000	25.000	98.000	6.618	276200.000
4	17.810	33.830	8.000	488.000	76.000	25.000	25.000	10.157	477100.000
5	18.580	34.250	23.000	488.000	76.000	25.000	25.000	5.521	280800.000
6	18.490	34.210	25.000	488.000	76.000	25.000	25.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	51400.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10

Done!

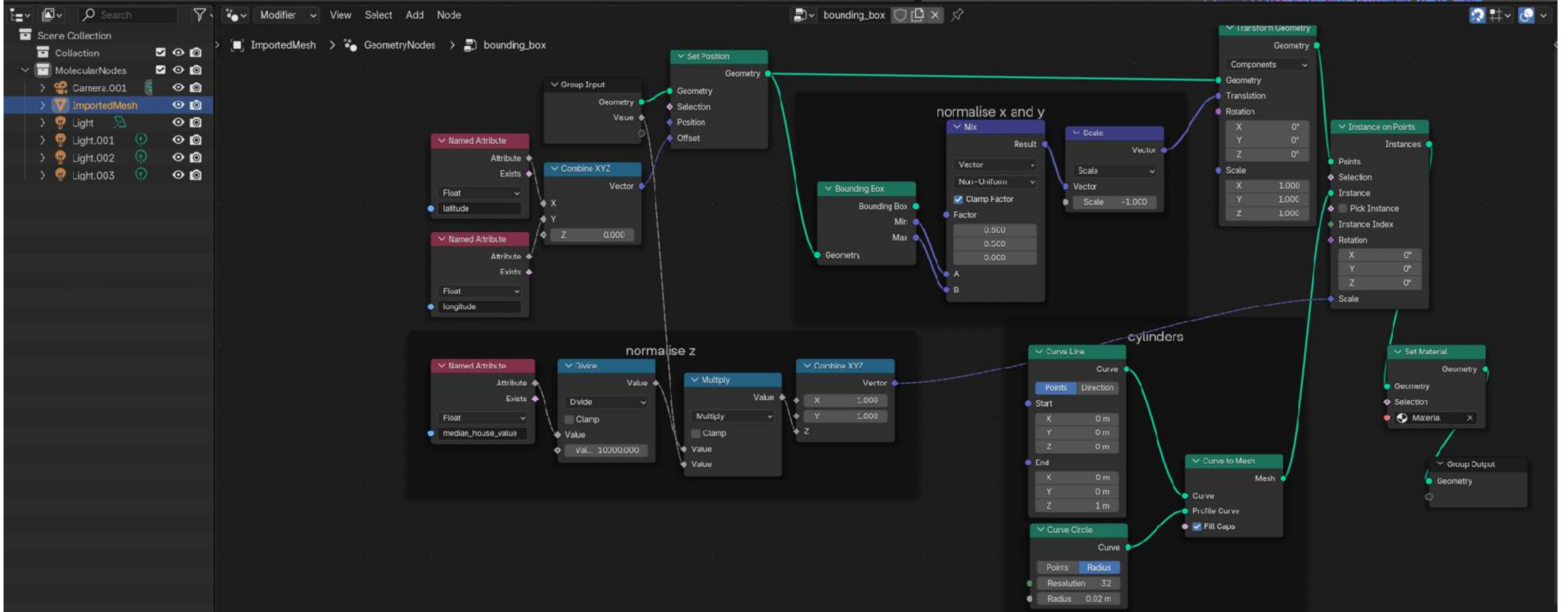


File Edit Render Window Help    Layout Shading **Geometry Nodes** Scripting +    Scene    ViewLayer

Original ImportedMesh

	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	20.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.691	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1294.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2669.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	769.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	51400.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

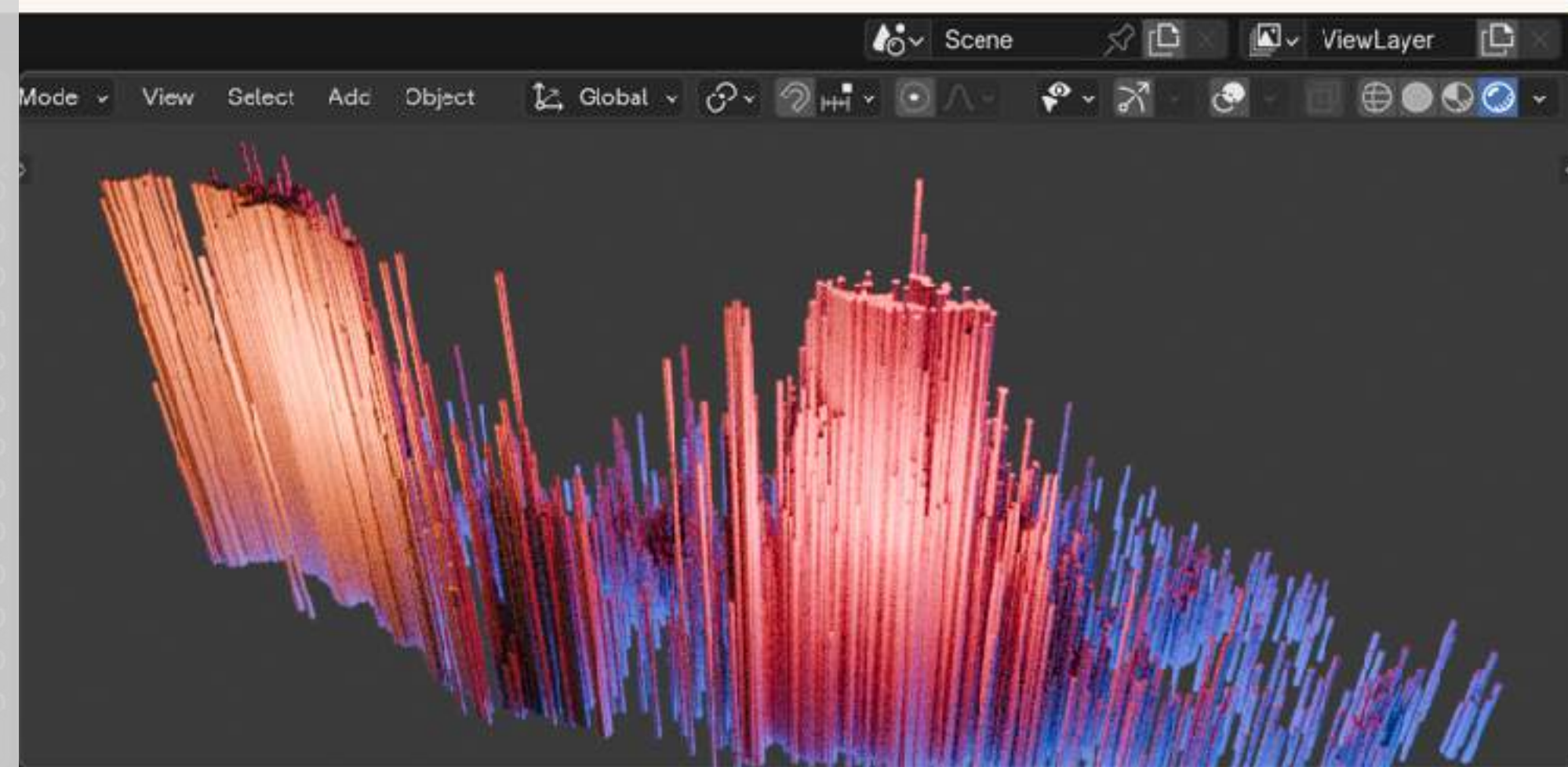
Rows: 12,590 | Columns: 10





	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	23.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.671	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1194.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2649.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	749.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.337	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	514000.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10

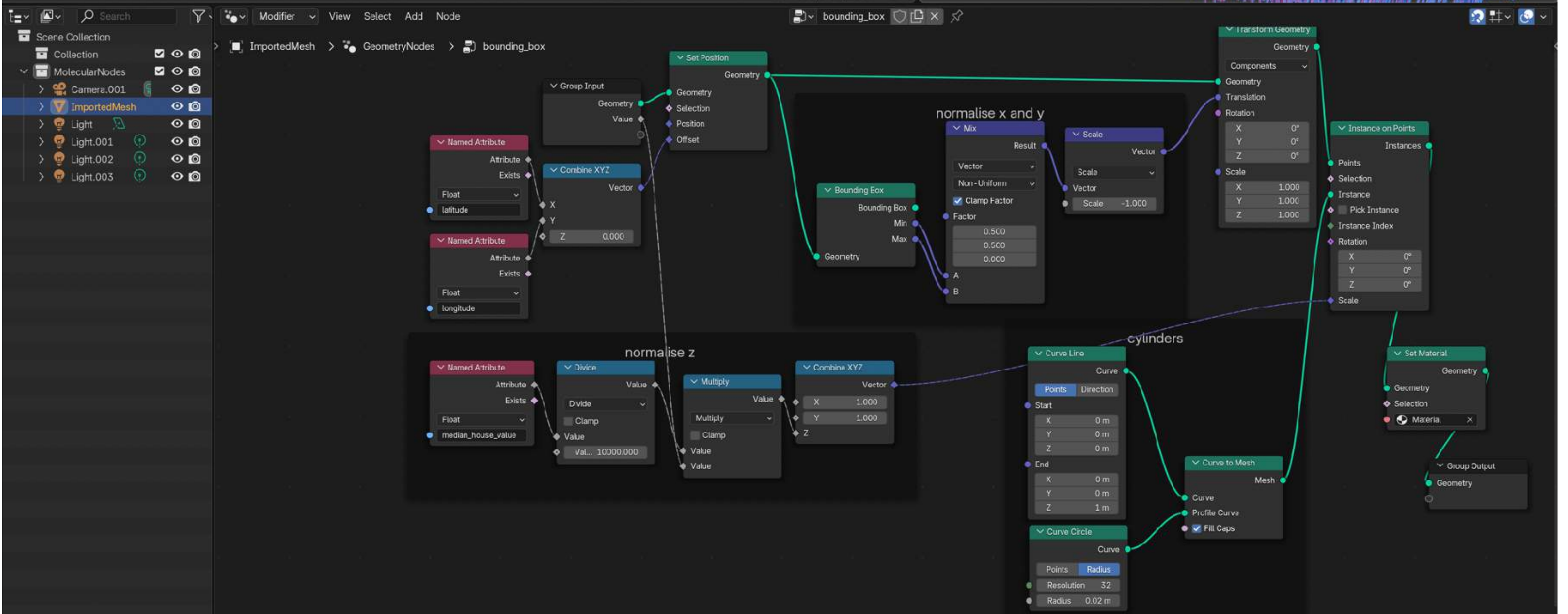


File Edit Render Window Help    Layout Shading Geometry Nodes Scripting +

Original ImportedMesh    Mode View Select Add Object    Global    ViewLayer

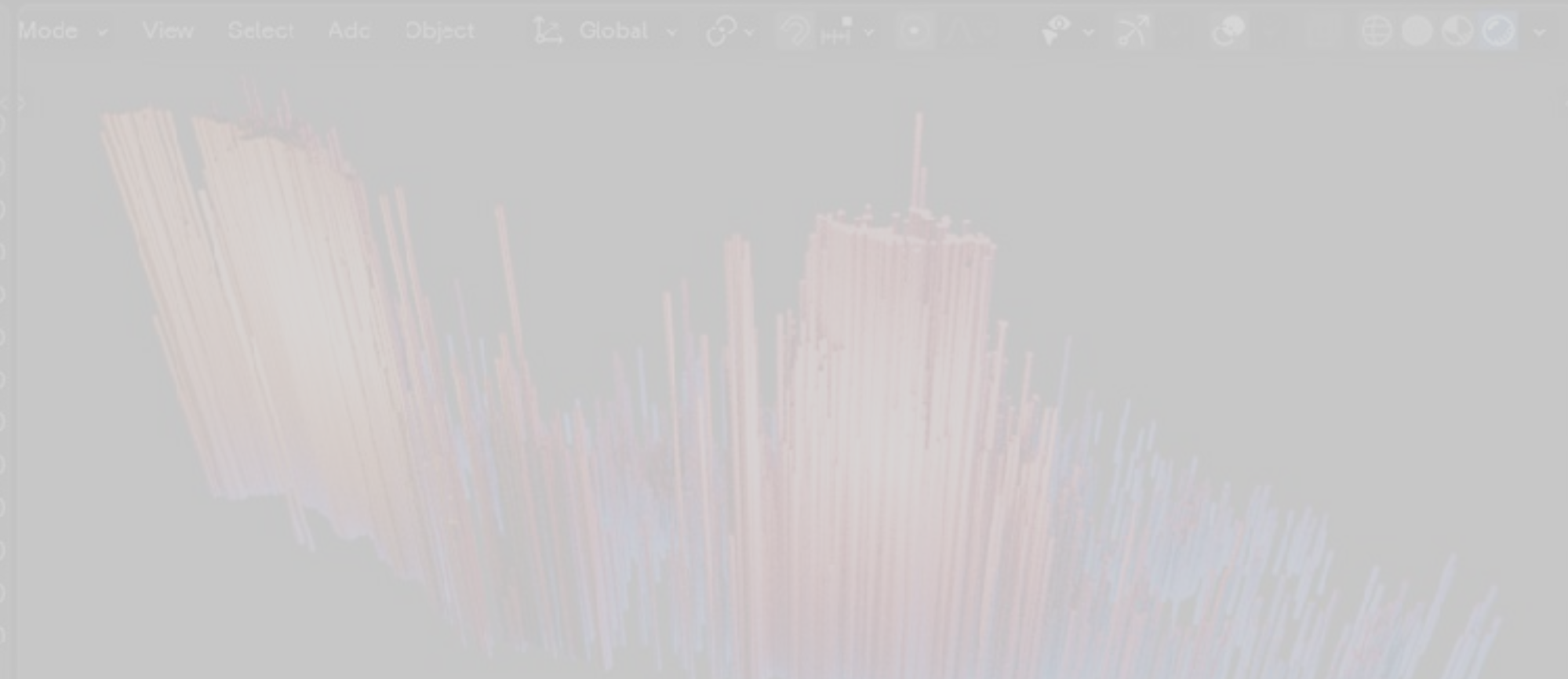
	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	23.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.671	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1194.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2649.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	749.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	51400.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10



	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	23.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.671	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1194.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	776200.000
4	17.810	33.830	8.000	7326.000	884.000	2649.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	749.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.337	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	514000.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10



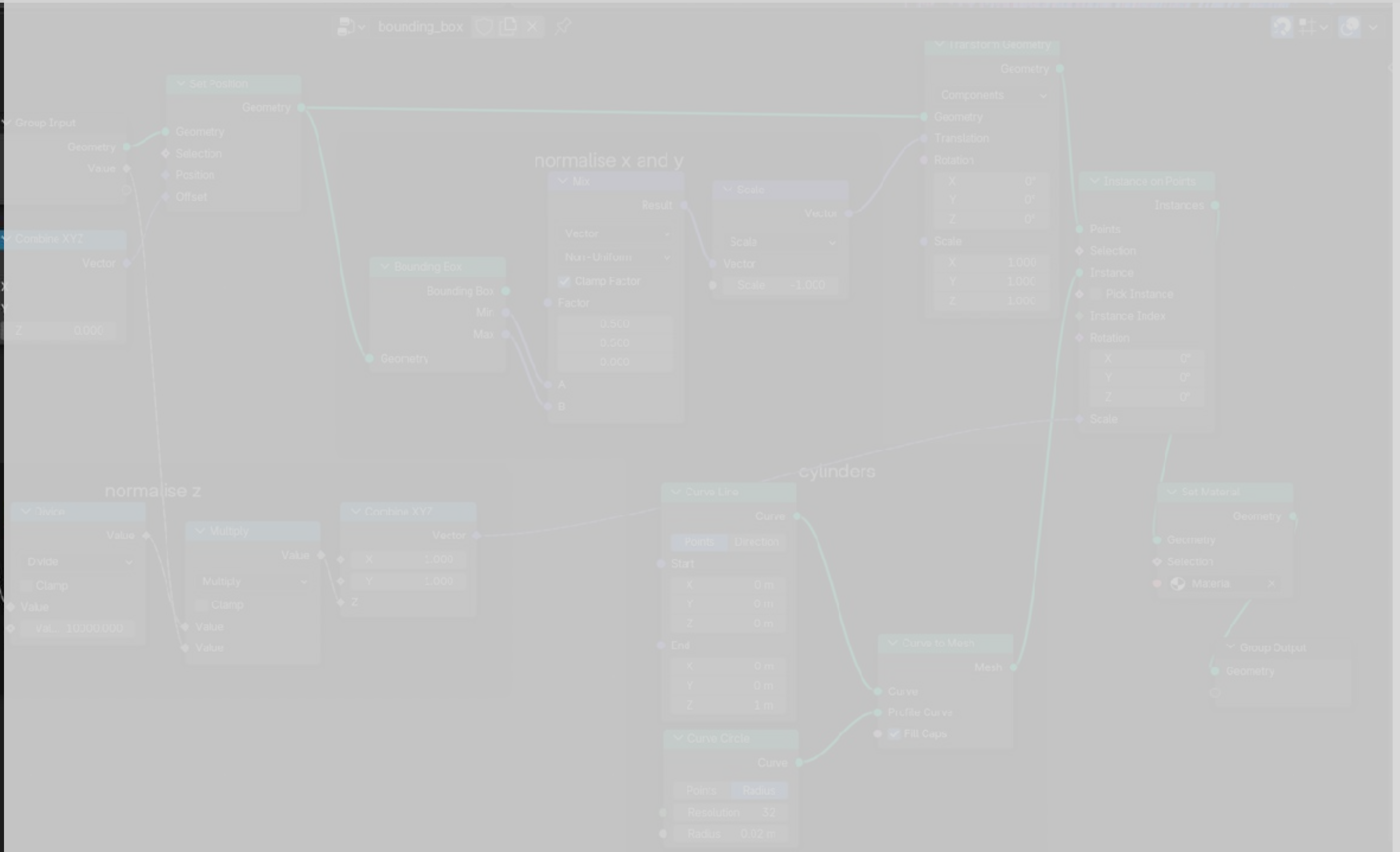
Scene Collection

- Collection
- MolecularNodes
- Camera.001
- ImportedMesh
- Light.001
- Light.002
- Light.003

ImportedMesh > GeometryNodes > bounding\_box

Named Attribute

- Attribute: latitude
- Attribute: longitude
- Attribute: median\_house\_value



Blender 2.80 Geometry Nodes interface showing a data table and a 3D visualization of a point cloud.

	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	20.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.691	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1294.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2669.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	769.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	514000.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	924000.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	1036000.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	1897000.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	3994000.000

Rows: 12,590 | Columns: 10

Blender 2.80 Geometry Nodes editor showing a node tree for creating cylinders from a point cloud. A red arrow points from the 'latitude' attribute in the table to the 'Named Attribute' node in the node tree.

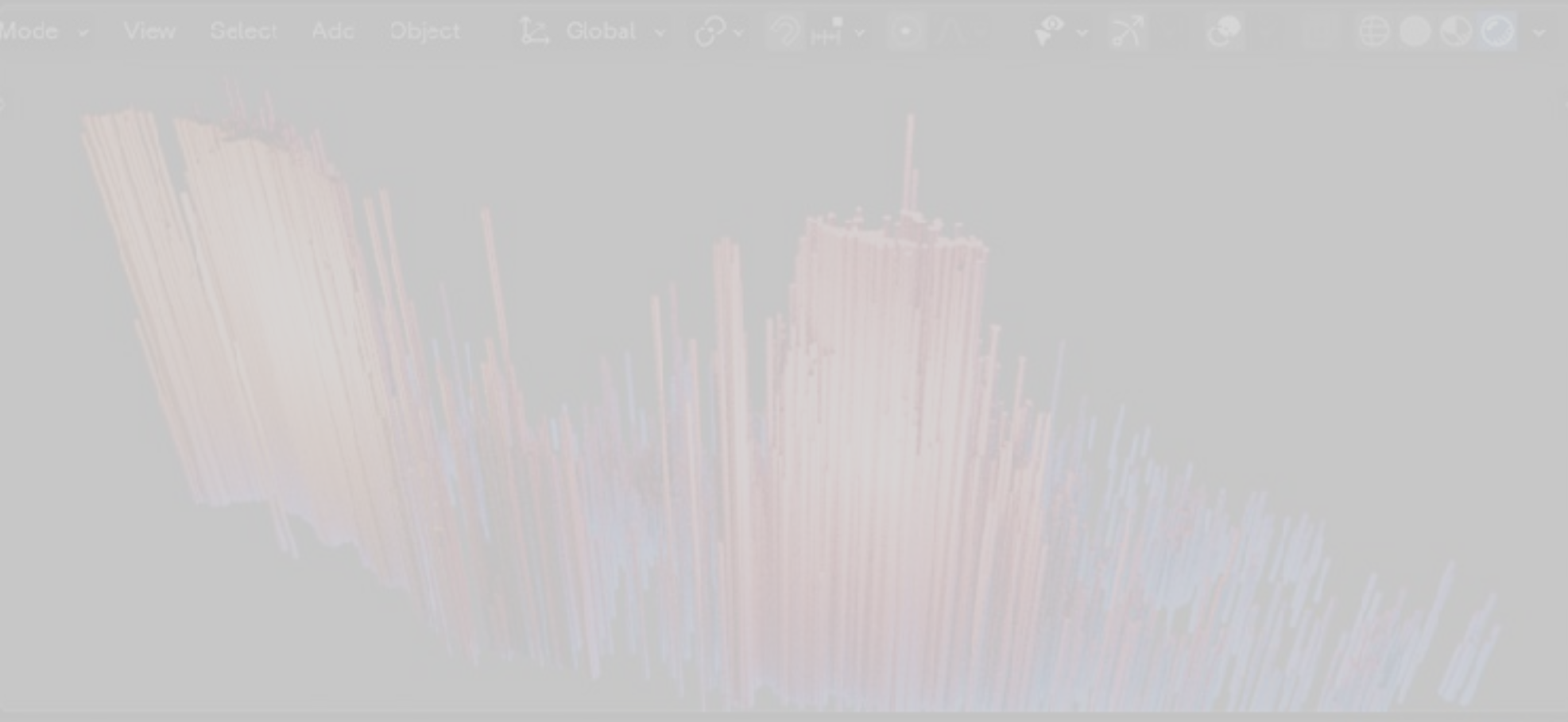
**Node Tree:**

- Group Input
- Named Attribute (Attribute: latitude)
- Combine XYZ (Vector: latitude, Z: 0.000)
- Set Position (Geometry)
- Bounding Box (Geometry)
- normalise x and y (Mix, Scale)
- normalise z (Divide, Multiply, Clamp)
- Curve Line (Curve)
- Curve Circle (Curve)
- Curve to Mesh (Mesh)
- Instance on Points (Instances)
- Set Material (Material)
- Group Output (Geometry)

**3D Viewport:** Shows a visualization of the resulting cylinders, colored by the 'latitude' attribute, with a bounding box overlaid.

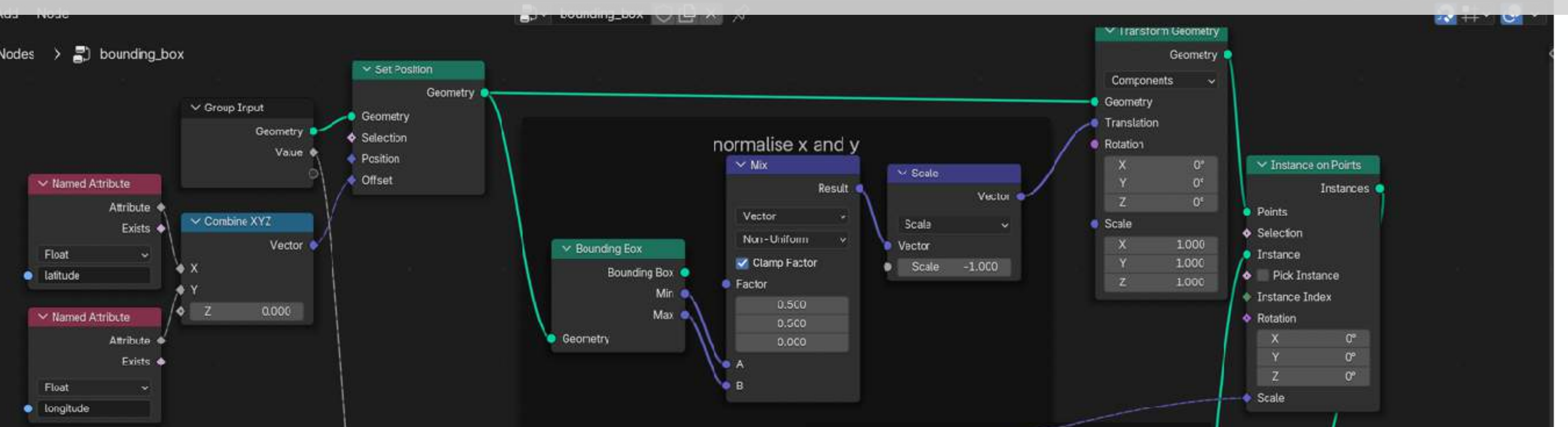
	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	23.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.671	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1194.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2649.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	749.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.357	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	514000.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	924000.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	1036000.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	1897000.000
12	19.910	36.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	3994000.000

Rows: 12,590 | Columns: 10



Scene Collection

- Collection
- MolecularNodes
- Camera.001
- ImportedMesh
- Light.001
- Light.002
- Light.003

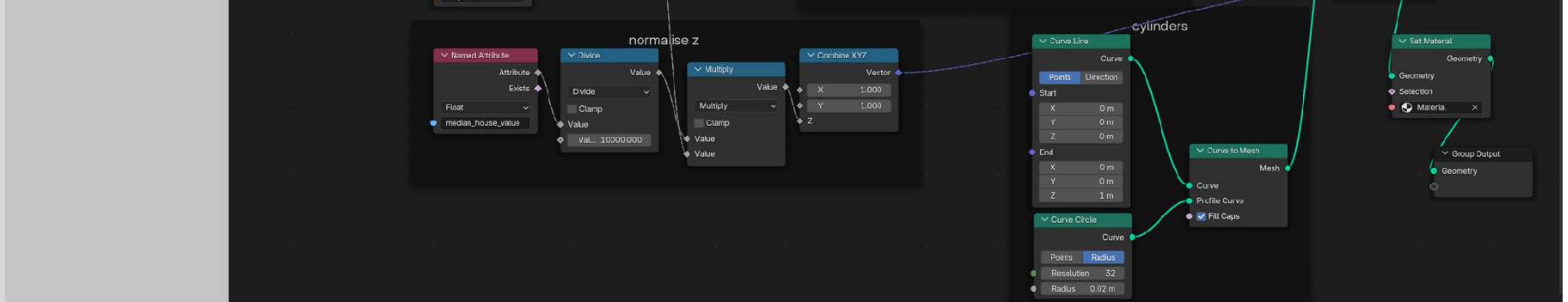
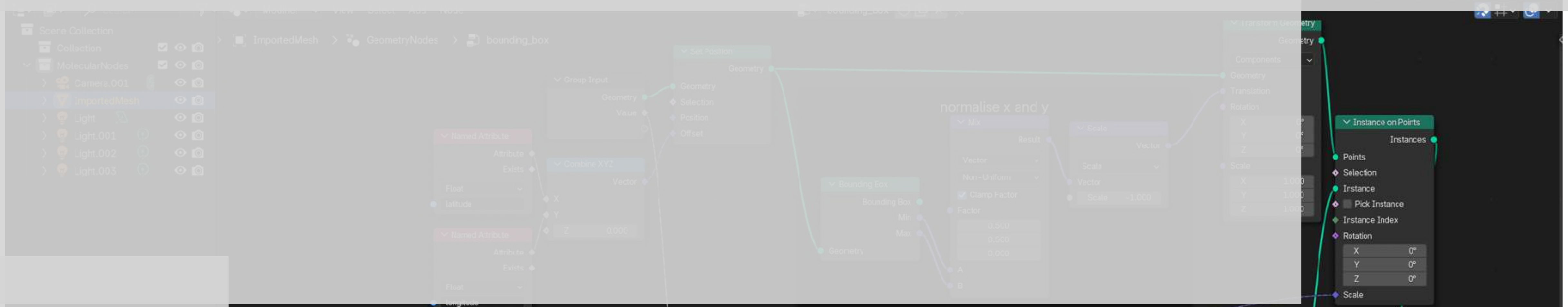


File Edit Render Window Help    Layout Shading Geometry Nodes Scripting +    Mode View Select Add Object Global ViewLayer

Original ImportedMesh

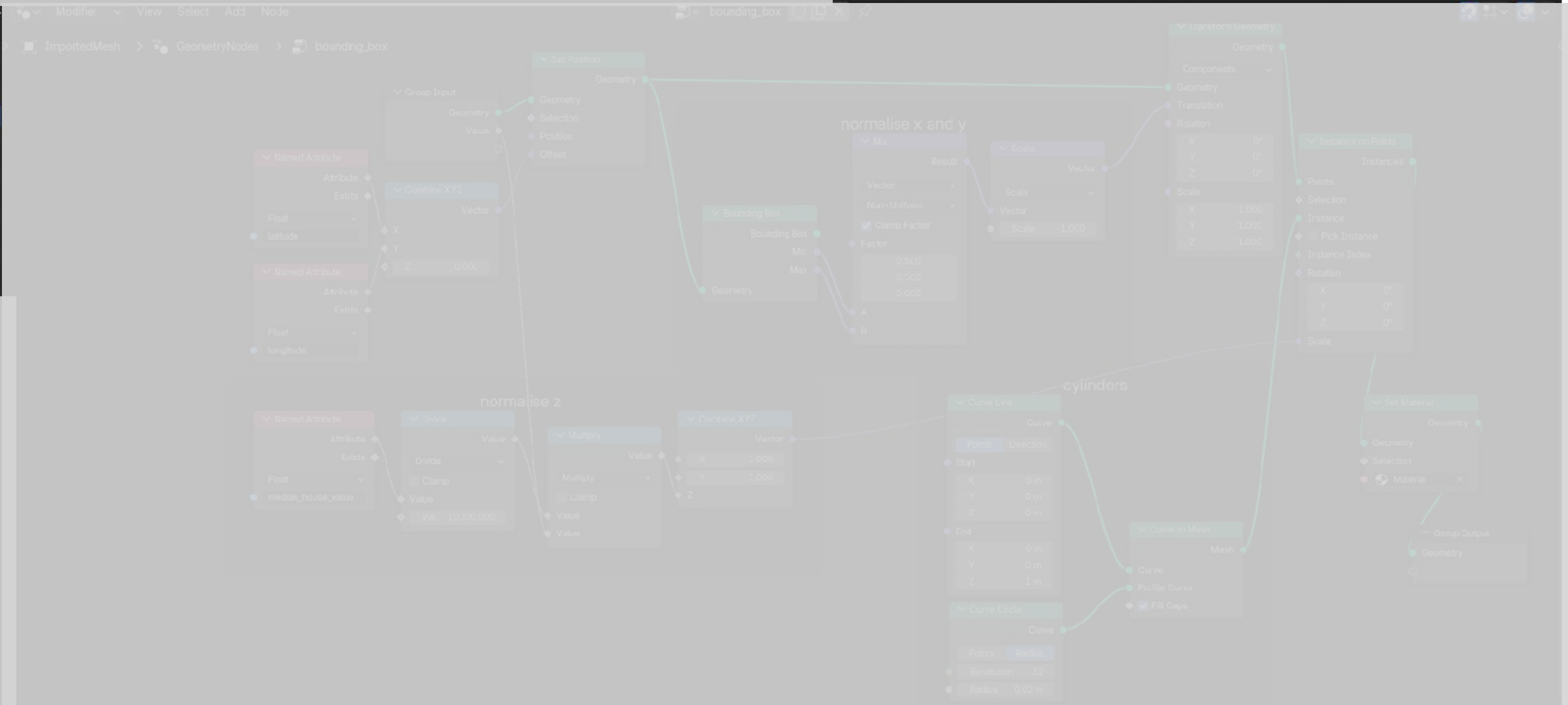
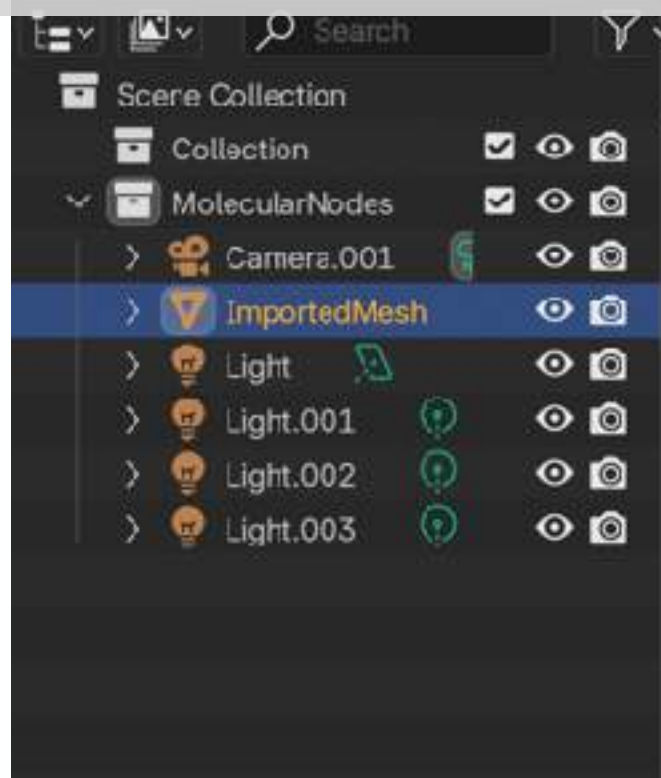
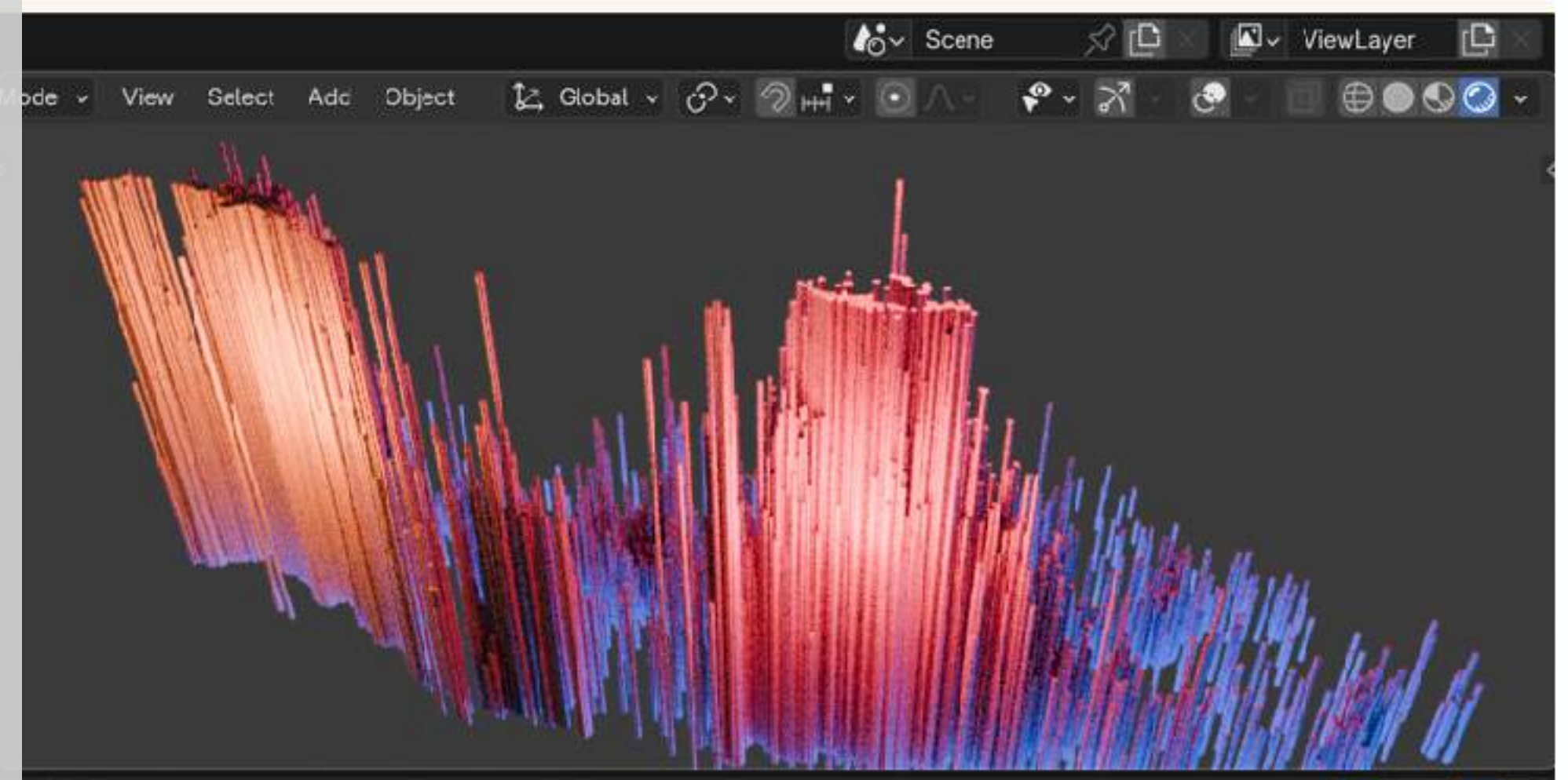
	gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	23.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.671	278800.000
1	22.420	37.600	34.000	3562.000	545.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1194.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2649.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	749.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.337	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	514000.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	23.920	35.400	23.000	2059.000	354.000	636.000	278.000	3.671	278800.000
1	22.420	37.600	34.000	3562.000	565.000	1542.000	563.000	5.878	405100.000
2	19.770	34.440	24.000	5652.000	1313.000	2312.000	1194.000	2.472	295300.000
3	21.900	37.450	18.000	4900.000	814.000	2984.000	758.000	6.618	276200.000
4	17.810	33.830	8.000	7326.000	884.000	2649.000	798.000	10.157	477100.000
5	18.580	34.250	23.000	4883.000	749.000	2119.000	725.000	5.521	280800.000
6	18.490	34.210	25.000	1131.000	449.000	746.000	420.000	1.337	225000.000
7	18.970	36.060	26.000	1289.000	262.000	1100.000	244.000	1.975	51400.000
8	17.900	36.950	19.000	99.000	26.000	51.000	22.000	1.729	137500.000
9	17.200	33.700	23.000	6323.000	1196.000	1984.000	1124.000	2.328	92400.000
10	24.300	41.840	17.000	2677.000	531.000	1244.000	456.000	3.031	103600.000
11	22.010	38.350	18.000	4486.000	723.000	1600.000	697.000	3.865	189700.000
12	18.840	34.160	18.000	6075.000	1056.000	2571.000	1018.000	5.220	399400.000

Rows: 12,590 | Columns: 10



# Time Series



1,000,000 inhabitants



**United Nations**  
Population Division  
Department of Economic and Social Affairs

World Urbanization Prospects: The 2018 Revision

File 22: Annual Population of Urban Agglomerations with 300,000 Inhabitants or More in 2018, by Country, 1950-2035 (thousands)

POP/DB/WUP/Rev.2018/1/F22

Copyright © 2018 by United Nations, made available under a Creative Commons license CC BY 3.0 IGO: <http://creativecommons.org/licenses/by/3.0/igo/>  
Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2018). World Urbanization Prospects: The 2018 Revision. Online Edition.

Index	Country Code	Country or area	City Code	Urban Agglomeration	Note	Latitude	Longitude	Annual Population of Urban Agglomerations with 300,000 Inhabitants or More in 2018, 1950-2035 (thousands)										
								1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
1	4	Afghanistan	20001	Herat		34.3482	62.1997	82	83	84	84	85	86	86	87	88	88	89
2	4	Afghanistan	20002	Kabul		34.5289	69.1725	171	180	189	199	210	221	232	245	257	271	285
3	4	Afghanistan	20003	Kandahar		31.6133	65.7101	82	84	85	87	88	90	91	93	95	96	98
4	4	Afghanistan	20004	Mazar-e Sharif		36.7090	67.1109	30	31	33	34	38	37	39	40	42	44	46
5	8	Albania	20005	Tiranë (Tirana)		41.3275	19.8139	65	89	93	97	102	107	112	117	123	129	135
6	12	Algeria	20009	Annaba		36.9000	7.7657	100	103	106	109	113	116	120	124	128	132	136
7	12	Algeria	20011	Batna		35.5660	5.1741	13	14	16	17	19	21	23	25	27	30	33
8	12	Algeria	20015	Blida		36.4808	2.8319	34	36	38	41	43	46	49	52	55	59	62
9	12	Algeria	20006	El Djazair (Algiers)	1	36.7825	3.0420	516	534	552	570	589	623	666	712	762	815	872
10	12	Algeria	20020	El Djelfa		34.6728	3.2630	10	11	12	12	13	14	15	16	17	18	18
11	12	Algeria	20019	Qacentina		36.3549	5.6073	68	94	100	107	114	122	130	139	148	158	169
12	12	Algeria	20037	Sétif		36.1911	5.4137	41	43	46	48	50	53	56	58	61	64	67
13	12	Algeria	20035	Wahran (Oran)		35.6911	-0.6417	269	272	276	279	283	286	290	294	297	301	305
14	24	Angola	895000001	Benguela	2	-12.5760	13.4050	14	15	16	16	17	18	19	20	21	22	23
15	24	Angola	204043	Caluanda	3	-5.5500	12.2000	5	5	5	5	5	5	5	5	5	5	5
16	24	Angola	895000011	Cuito		-12.3030	15.9330	9	9	9	10	10	10	11	11	12	12	13
17	24	Angola	20050	Huambo	4	-12.7761	15.7392	16	17	18	20	22	24	26	28	31	34	37
18	24	Angola	895000000	Lobito	5	-12.3640	13.5360	23	25	27	29	31	33	36	39	42	45	48
19	24	Angola	20049	Luanca	6	-8.8368	13.2343	138	145	152	159	166	174	182	191	200	210	219
20	24	Angola	204044	Lubango	7	-14.9172	13.4925	12	12	12	12	13	13	13	14	14	14	15
21	24	Angola	895000004	Malanje	8	-9.5400	15.3410	9	10	11	11	12	13	14	15	16	17	18
22	24	Angola	895000012	Uige		-7.1069	15.4405	3	3	4	4	4	4	5	5	6	6	6
23	32	Argentina	20057	Bahia Blanca		-38.7196	-62.2724	116	117	118	119	120	121	122	123	124	125	126
24	32	Argentina	20058	Buenos Aires	9	-34.6051	-58.4004	5 166	5 307	5 452	5 601	5 753	5 910	6 072	6 237	6 407	6 582	6 762
25	32	Argentina	20059	Córdoba	10	-31.4135	-64.1311	429	444	460	476	493	510	528	545	565	585	605
26	32	Argentina	20063	Corrientes	11	-27.4606	-58.8341	64	67	70	73	76	79	82	85	89	93	97
27	32	Argentina	20065	La Plata	12	-34.6215	-57.9545	300	308	318	327	337	347	357	368	378	390	401
28	32	Argentina	20067	Mar Del Plata		-38.0023	-57.5575	132	139	146	152	159	166	174	182	191	200	209
29	32	Argentina	20068	Mendoza	13	-32.8508	-68.8272	246	253	260	268	276	284	292	301	310	319	328
30	32	Argentina	20070	Neuquén-Pollih-Cipolletti		-38.9514	-68.0648	14	15	17	19	20	23	25	27	30	33	37
31	32	Argentina	20072	Posadas		-27.3671	-55.8951	46	48	50	53	55	56	61	63	67	70	73
32	32	Argentina	20073	Resistencia	14	-27.4606	-58.9830	73	75	78	81	84	87	90	94	97	101	105
33	32	Argentina	20075	Rosario	15	-32.6468	-60.6393	554	565	576	587	599	610	622	634	646	659	671
34	32	Argentina	20076	Salta		-24.7859	-65.4117	77	81	84	87	91	95	99	103	108	112	117
35	32	Argentina	20078	San Juan	16	-31.5375	-68.5354	108	112	115	119	123	127	131	135	140	144	149
36	32	Argentina	20081	San Miguel de Tucumán	17	-26.8241	-65.2225	224	231	237	245	252	260	267	275	284	292	301
37	32	Argentina	20084	San Salvador de Jujuy	18	-24.1046	-65.2971	34	35	36	37	38	39	40	41	42	43	44
38	32	Argentina	20085	Santa Fe	19	-31.6327	-60.6983	214	219	224	229	234	239	244	249	255	260	266
39	32	Argentina	20086	Santiago Del Estero	20	-27.7951	-64.2615	83	85	86	88	90	93	95	97	99	101	104
40	51	Armenia	20140	Yerevan		40.1820	44.5145	341	358	375	393	412	431	452	474	497	518	538
41	36	Australia	205171	Adelaide	21	-34.8287	138.5959	429	443	456	470	484	497	511	525	541	556	572
42	36	Australia	205170	Brisbane	21	-27.4679	153.0231	442	456	471	486	502	516	534	550	567	585	603
43	36	Australia	205176	Canberra	21	-35.2835	149.1281	20	22	24	26	28	31	36	38	42	46	51
44	36	Australia	895000015	Central Coast		-33.4200	151.3958	72	74	76	78	80	82	84	87	89	92	94
45	36	Australia	20093	Gold Coast-Tweed Head	21	-28.0003	153.4309	29	31	32	34	36	38	40	42	44	47	50
46	36	Australia	205168	Melbourne	21	-37.8140	144.9633	1 332	1 377	1 424	1 473	1 524	1 574	1 626	1 680	1 735	1 792	1 851
47	36	Australia	205177	Newcastle-Maitland	21	-32.5249	151.7409	161	166	170	174	179	182	186	191	195	199	204
48	36	Australia	205172	Perth	21	-31.6478	115.8525	311	320	330	339	349	358	368	378	388	398	409

Sheet Name

Data

Background

Duplicate Sheet

Delete Sheet

```

from pathlib import Path
import polars as pl
file_path = Path.home() / "projects" / "animation_earth_population" / "WUP2018-F22-Cities_Over_300K_Annual.xls"
file_path.exists()

df = pl.read_excel(
    file_path, # Use the Path object here
    read_options={"skip_rows": 16}, # Skip rows up to the header
    has_header=False, # Indicate no header is currently in use
)

# Use the first row as headers and drop it
df.columns = [str(x) for x in df.row(0)]
df = df[1:]

# Rename floatlike column names such as "1950.0" -> "1950"
df.columns = [col[:-2] if col.endswith(".0") else col for col in df.columns]

# Cast Index, Country Code, City Code to integers; Latitude & Longitude to strings
df = df.with_columns(
    pl.col("Index").cast(pl.Int64),
    pl.col("Country Code").cast(pl.Int64),
    pl.col("City Code").cast(pl.Int64),
    pl.col("Latitude").cast(pl.Float64),
    pl.col("Longitude").cast(pl.Float64),
)

df.head()

```

[36] ✓ 0.0s Open 'df' in Data Wrangler Python

shape: (5, 94)

Index	Country Code	Country or area	City Code	Urban Agglomeration	Note	Latitude	Longitude	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962
i64	i64	str	i64	str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
1	4	"Afghanistan"	20001	"Herat"	null	34.34817	62.19967	82.468	83.114	83.767	84.422	85.084	85.751	86.424	87.101	87.763	88.472	89.166	89.664	90.000
2	4	"Afghanistan"	20002	"Kabul"	null	34.528887	69.17246	170.784	179.779	189.26	199.213	209.705	220.749	232.391	244.613	257.496	271.057	285.352	300.359	316.000
3	4	"Afghanistan"	20003	"Kandahar"	null	31.61332	65.71013	82.199	83.663	85.155	86.67	88.214	89.786	91.387	93.012	94.669	96.356	98.074	99.818	101.000
4	4	"Afghanistan"	20004	"Mazar-e Sharif"	null	36.70904	67.11087	30.0	31.308	32.676	34.099	35.586	37.139	38.761	40.449	42.213	44.054	45.979	47.931	50.000
5	8	"Albania"	20005	"Tiranë (Tirana)"	null	41.3275	19.81889	84.513	88.868	93.08	97.481	102.097	106.932	112.001	117.295	122.847	128.662	134.761	137.714	139.000

```

from csv_importer.parsers import polars_df_to_bob

```

[37] ✓ 0.0s Python

```

from io import StringIO
import polars as pl
bob = polars_df_to_bob(df, name="World")
bob.name

```

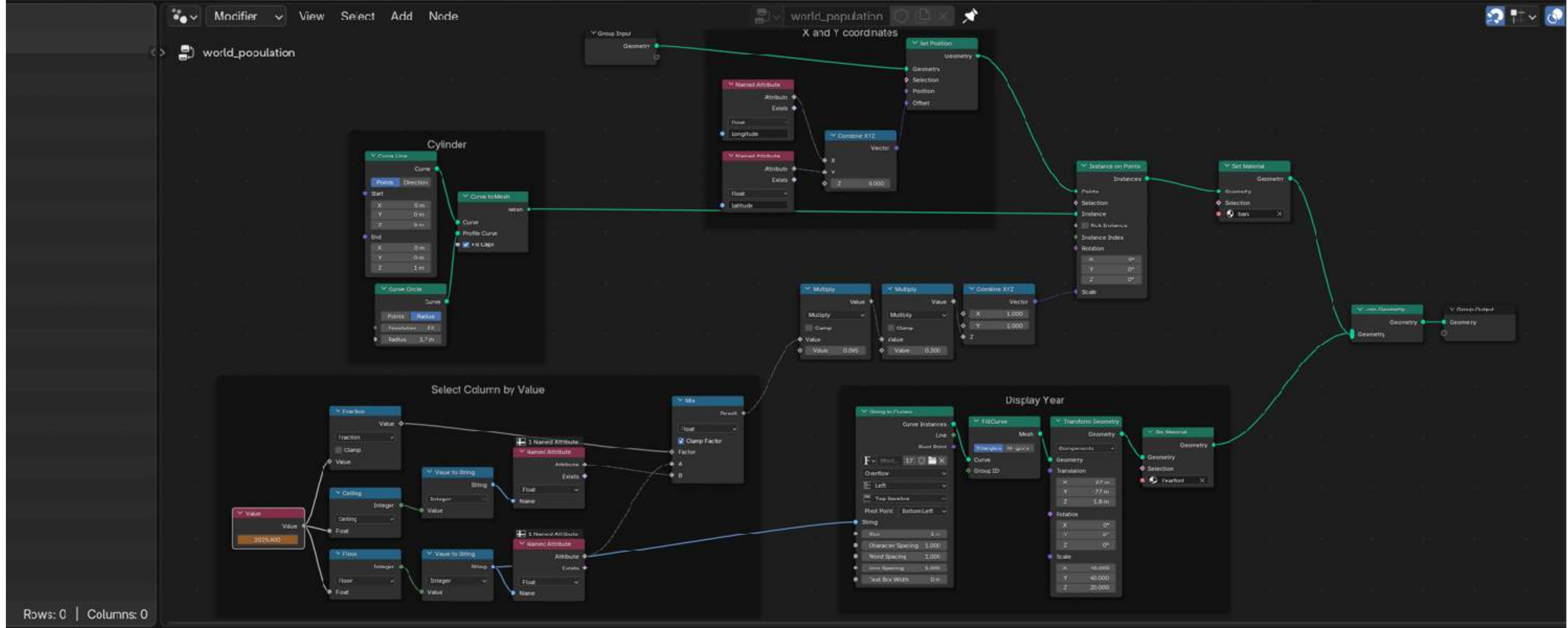
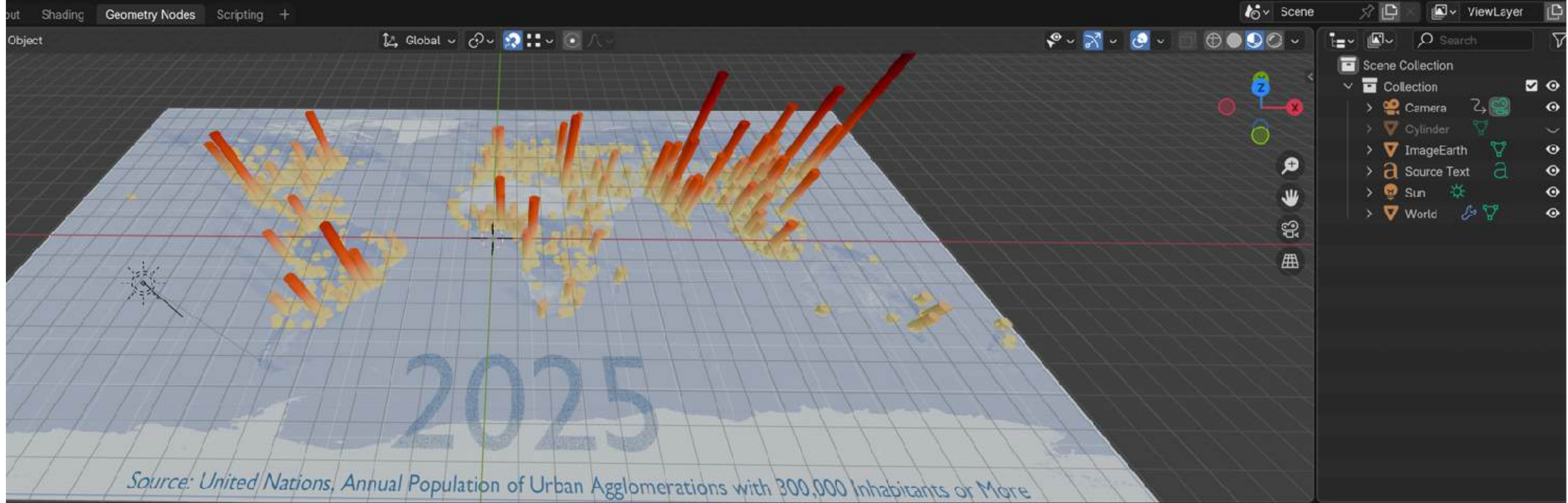
[38] ✓ 0.1s Python

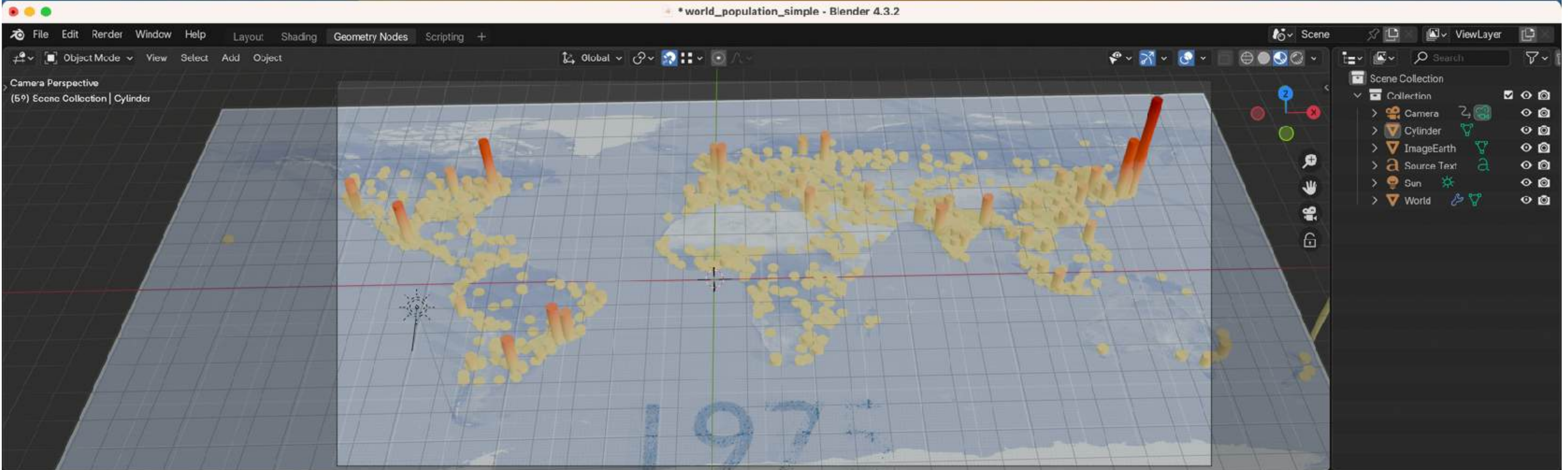
'World'

\* (Unsaved) - Blender 4.3.2

Texture Paint Shading Animation Rendering Compositing **Geometry Nodes** Scripting +

Index	Country Code	Country or area	City Code	Urban Agglomeration	Note	Latitude	Longitude	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962
000	0.000	34.348	62.200	82.468	83.114	83.767	84.422	85.084	85.751	86.424	87.101	87.763	88.472	89.166	89.664	90.000	90.000	90.000	90.000	90.000
000	0.000	34.529	69.172	170.784	179.779	189.260	199.213	209.705	220.749	232.391	244.613	257.496	271.057	285.352	300.359	316.000	332.000	348.000	364.000	380.000
000	0.000	31.613	65.710	82.199	83.663	85.155	86.670	88.214	89.786	91.387	93.012	94.669	96.356	98.074	99.818	101.000	102.000	103.000	104.000	105.000
000	0.000	36.709	67.111	30.000	31.308	32.676	34.099	35.586	37.139	38.761	40.449	42.213	44.054	45.979	47.931	49.883	51.835	53.787	55.739	57.691
000	0.000	41.327	19.819	84.513	88.866	93.080	97.481	102.097	106.932	112.001	117.295	122.847	128.662	134.761	140.860	146.959	153.058	159.157	165.256	171.355
000	0.000	36.900	7.767	99.741	102.878	106.118	109.451	112.893	116.335	119.777	123.220	126.662	130.104	133.546	136.988	140.430	143.872	147.314	150.756	154.198
000	0.000	35.556	6.174	13.046	14.306	15.689	17.201	18.862	20.614	22.466	24.418	26.470	28.622	30.874	33.226	35.678	38.130	40.582	43.034	45.486
000	0.000	36.481	2.832	33.970	36.092	38.349	40.742	43.286	45.979	48.822	51.815	54.858	57.951	61.094	64.287	67.530	70.823	74.166	77.559	80.902
000	0.000	36.752	3.042	516.450	533.740	551.632	570.074	589.158	608.886	629.266	650.294	671.970	694.294	717.266	740.886	765.058	789.790	815.082	840.934	867.346
000	0.000	34.673	3.263	10.450	11.061	11.708	12.392	13.117	13.881	14.684	15.526	16.407	17.327	18.286	19.284	20.321	21.397	22.512	23.666	24.859
000	0.000	36.355	6.607	88.001	93.924	100.254	106.992	114.193	121.814	129.815	138.156	146.807	155.748	164.969	174.450	184.181	194.162	204.393	214.874	225.605
000	0.000	36.191	5.414	41.289	43.327	45.468	47.709	50.063	52.531	55.114	57.812	60.625	63.553	66.596	69.754	73.027	76.415	79.918	83.536	87.269
000	0.000	35.691	-0.642	268.935	272.335	275.782	279.263	282.793	286.372	290.000	293.677	297.404	301.181	305.008	308.885	312.812	316.789	320.816	324.893	329.020
000	0.000	-12.576	13.405	14.364	15.042	15.752	16.493	17.270	18.083	18.932	19.816	20.734	21.686	22.672	23.692	24.745	25.831	26.949	28.099	29.281
000	0.000	-5.550	12.200	4.722	4.710	4.698	4.687	4.675	4.663	4.651	4.639	4.627	4.615	4.603	4.591	4.579	4.567	4.555	4.543	4.531
000	0.000	-12.383	16.933	8.632	8.970	9.322	9.686	10.066	10.461	10.871	11.296	11.735	12.188	12.655	13.136	13.631	14.140	14.663	15.200	15.751
000	0.000	-12.776	15.739	15.306	16.721	18.270	19.957	21.802	23.816	25.907	28.084	30.347	32.696	35.131	37.652	40.259	42.952	45.731	48.596	51.547
000	0.000	-12.364	13.536	23.027	24.801	26.714	28.769	30.986	33.366	35.919	38.556	41.277	44.082	46.971	49.944	53.001	56.142	59.367	62.676	66.069
000	0.000	-8.837	13.234	138.413	144.939	151.782	158.928	166.422	174.264	182.453	190.990	199.875	209.108	218.689	228.518	238.595	248.920	259.493	270.314	281.383
000	0.000	-14.917	13.493	11.555	11.846	12.146	12.453	12.767	13.088	13.415	13.748	14.087	14.431	14.780	15.134	15.493	15.857	16.226	16.599	16.976
000	0.000	-9.540	16.341	9.176	9.834	10.541	11.296	12.107	12.974	13.897	14.876	15.911	16.999	18.141	19.337	20.588	21.894	23.255	24.671	26.142
000	0.000	-7.107	15.441	3.201	3.414	3.642	3.884	4.143	4.417	4.706	5.009	5.326	5.657	6.001	6.358	6.728	7.111	7.506	7.913	8.332
000	0.000	-38.720	-62.272	115.752	116.774	117.807	118.846	119.895	120.954	122.023	123.101	124.188	125.284	126.389	127.503	128.626	129.758	130.899	132.049	133.208
000	0.000	-34.605	-58.400	5166.140	5307.065	5452.036	5600.553	5753.328	5910.259	6071.164	6235.953	6404.546	6576.863	6752.824	6932.350	7115.371	7300.817	7489.618	7681.704	7877.015
000	0.000	-31.413	-64.181	429.249	444.257	459.811	475.865	492.502	509.739	527.586	545.943	564.820	584.227	604.174	624.571	645.418	666.725	688.492	710.729	733.446

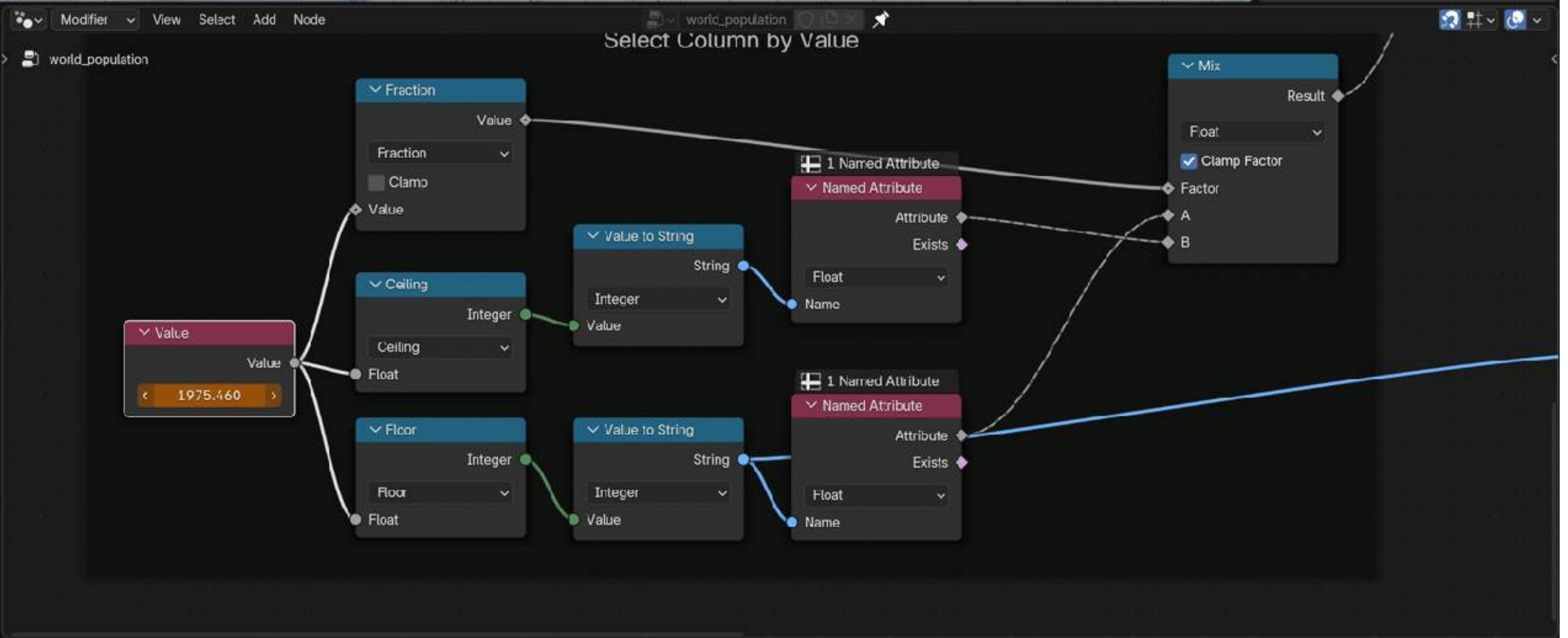


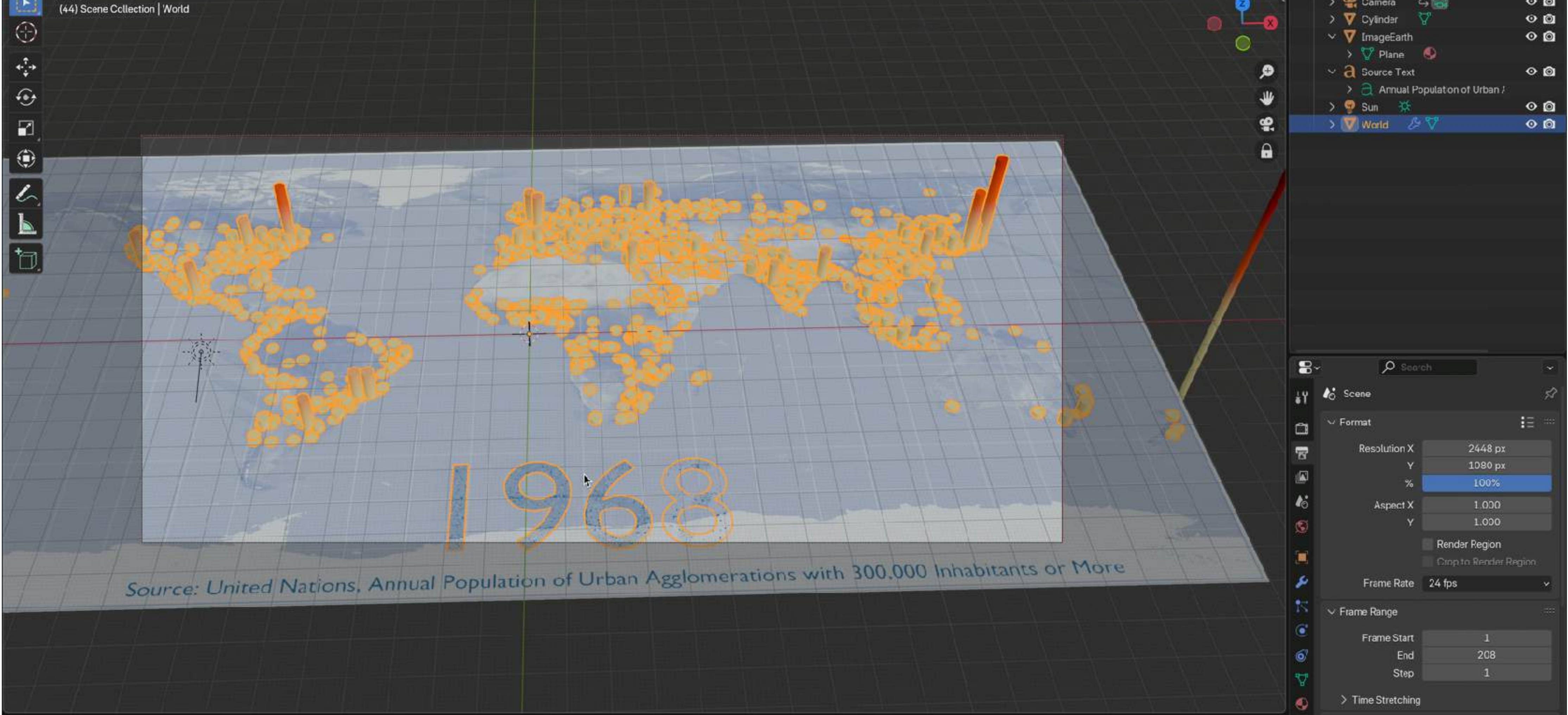


Original

position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	16	17	17	18	18	19	19	20	20	21	21	22	22	23	23
0	0.000	1.000	-1.000																													
1	0.000	1.000	1.000																													
2	0.195	0.981	-1.000																													
3	0.195	0.981	1.000																													
4	0.383	0.924	-1.000																													
5	0.383	0.924	1.000																													
6	0.556	0.831	-1.000																													
7	0.556	0.831	1.000																													
8	0.707	0.707	-1.000																													
9	0.707	0.707	1.000																													
10	0.831	0.556	-1.000																													
11	0.831	0.556	1.000																													
12	0.924	0.383	-1.000																													
13	0.924	0.383	1.000																													
14	0.981	0.195	-1.000																													
15	0.981	0.195	1.000																													
16	1.000	0.000	-1.000																													
17	1.000	0.000	1.000																													
18	0.981	-0.195	-1.000																													
19	0.981	-0.195	1.000																													
20	0.924	-0.383	-1.000																													
21	0.924	-0.383	1.000																													
22	0.831	-0.556	-1.000																													
23	0.831	-0.556	1.000																													

Rows: 64 | Columns: 1





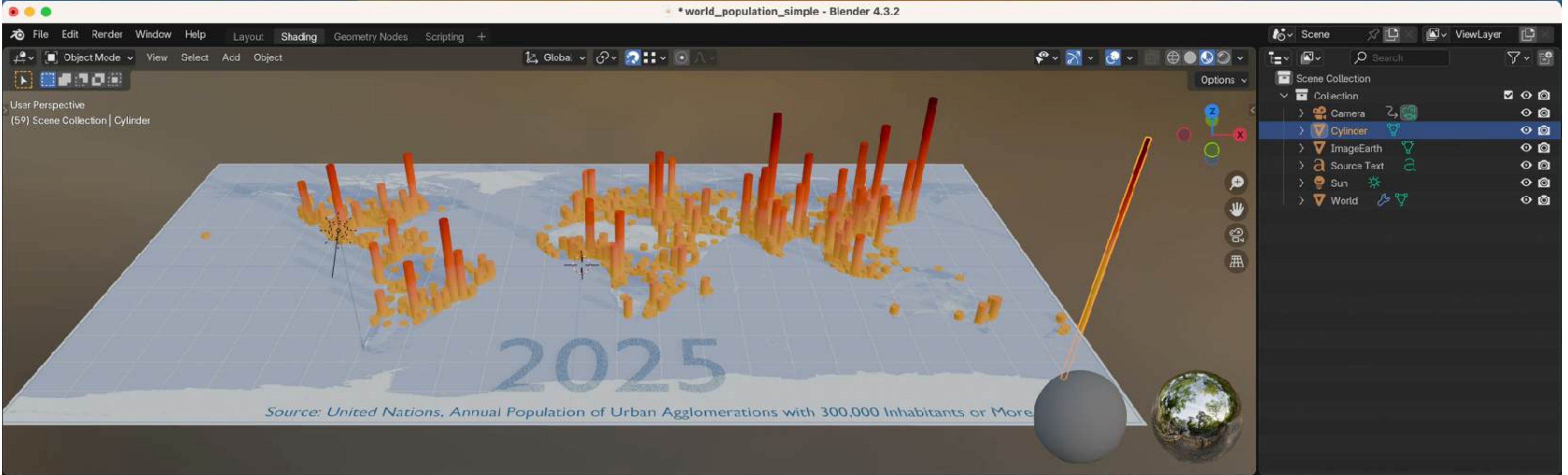
Playback | Keying | View | Marker | 44 | Start 1 | End 208

Summary | World | world\_population

Action | world\_populationAction | Manual Frame Range | Start 0.000 | End 0.000 | Cyclic: Animation

Output | /Users/jan-hendrik/Desktop/hi | Saving  File Extensions | Cache Result  | File Format PNG

Select | Rotate View | Object | Anim Player | 4.3.2



Object Mode | View | Select | Add | Node | Use Nodes | Slot 1 | bars | 3 | X |

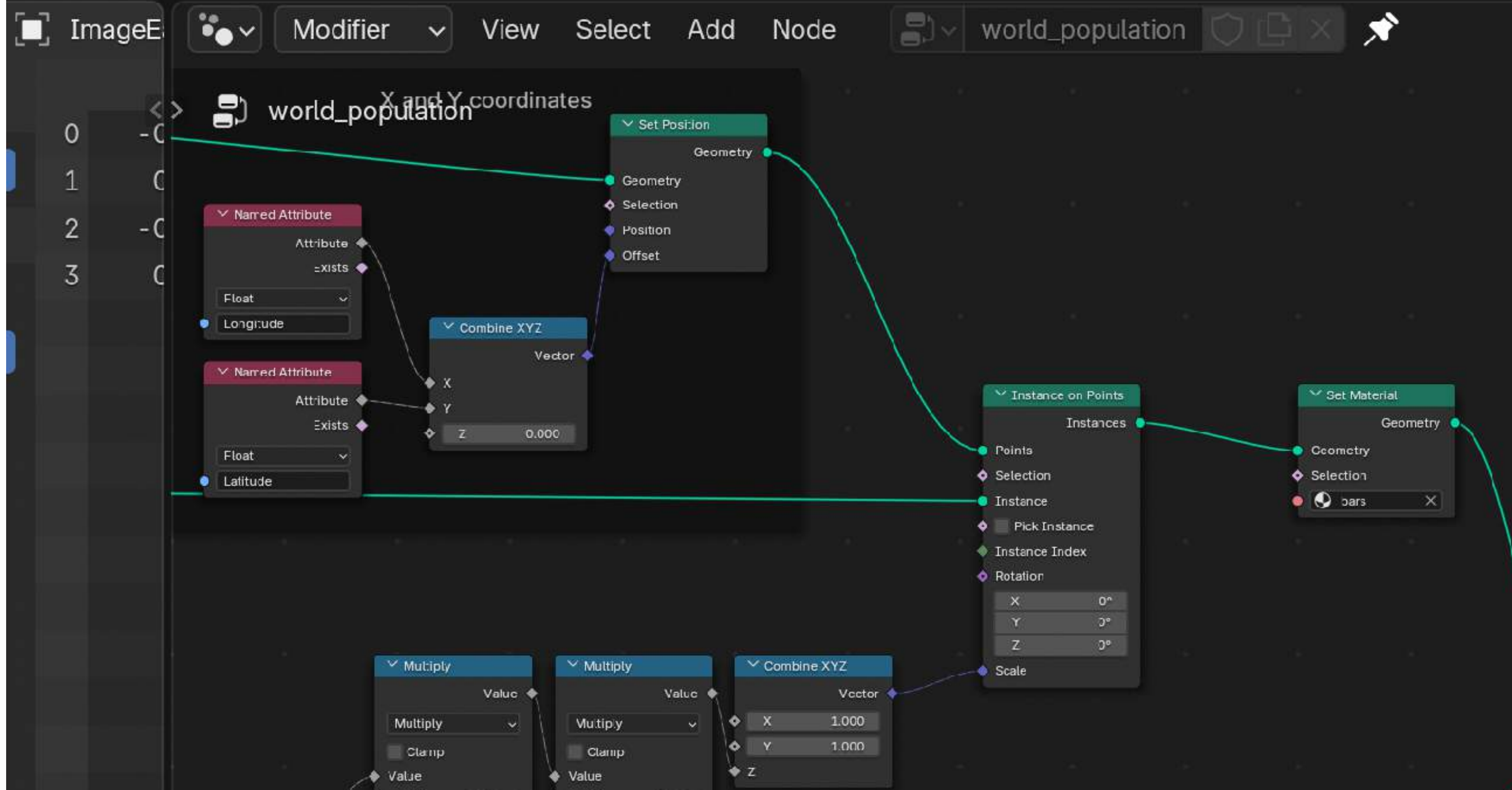
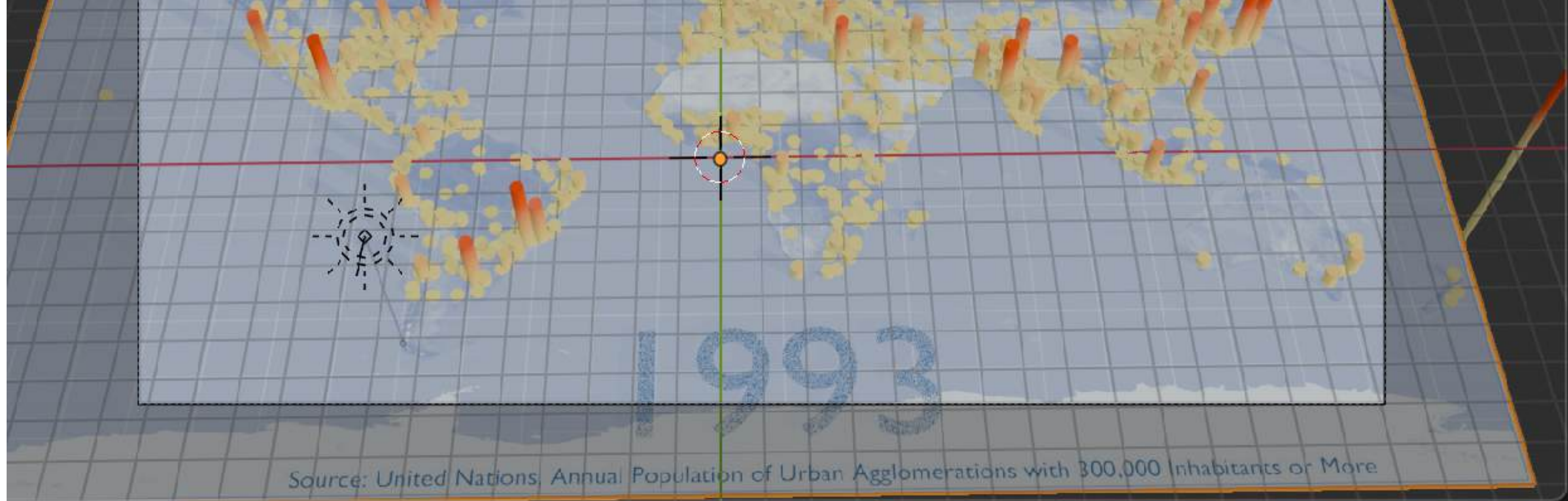
Cylinder > Cylinder > bars

Texture Coordinate -> Separate XYZ -> Color Ramp -> Principled BSDF -> Material Output

Material Output Settings:

- Format: PNG
- Color: RGB
- Color Depth: 8
- Compression: 15%
- Image Sequence: Overwrite
- Color Management: Override
- Display Device: Display P3
- View: Standard
- Look: None
- Exposure: 0.000
- Gamma: 1.000

4.3.2





```
string_to_label.ipynb
```

```
import bpy
blend_file_path = "string_to_label.blend"
bpy.ops.wm.open_mainfile(filepath=blend_file_path)

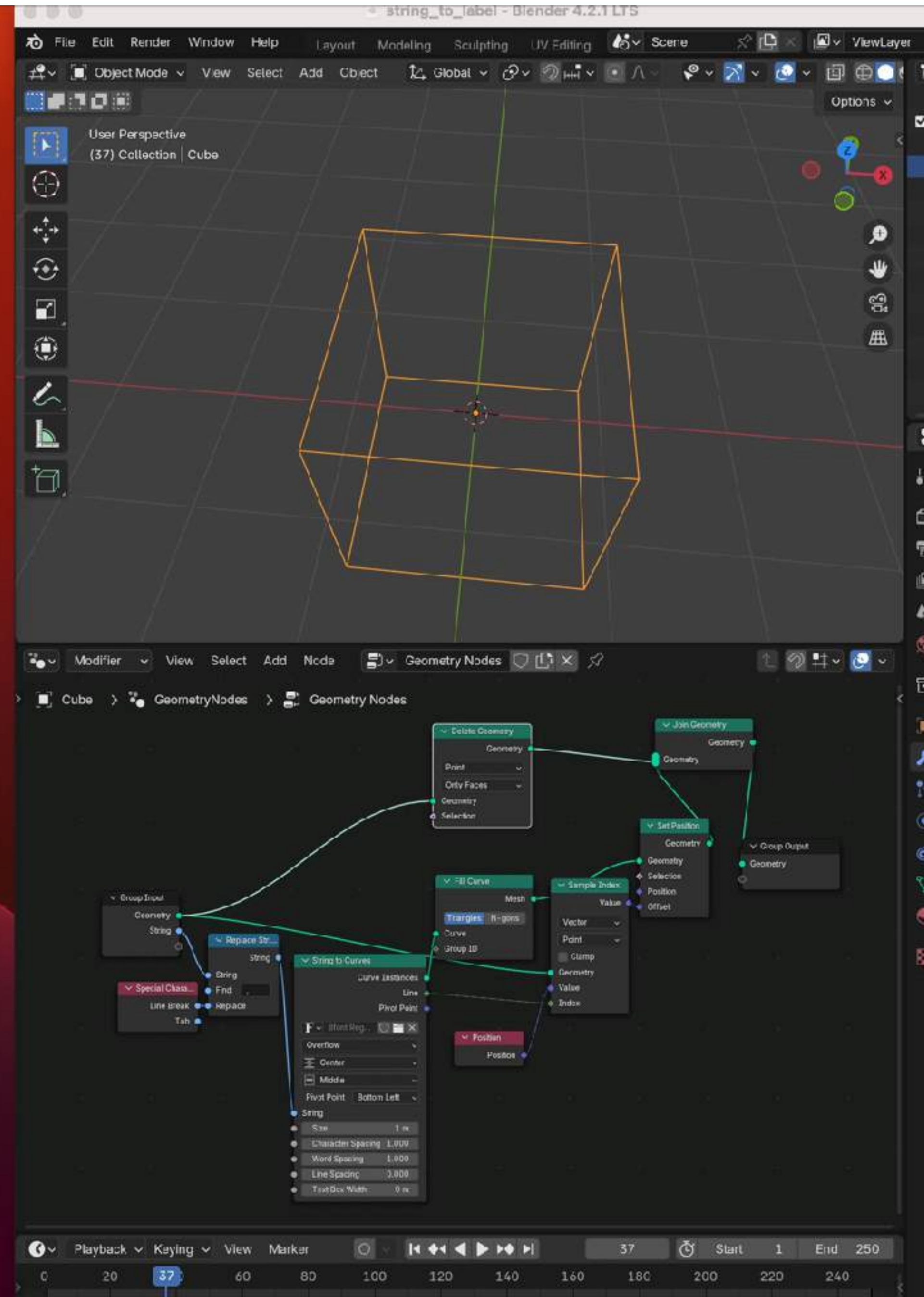
def set_labels(value_list):
    # Get the object and its Geometry Nodes modifier
    obj = bpy.data.objects.get("Cube")
    geo_modifier = obj.modifiers.get("GeometryNodes")
    new_lst = ','.join(value_list)
    print(new_lst)
    geo_modifier["Socket_2"] = new_lst
    obj.data.update()

# Example call with the list of values
set_labels([" "])
```

```
set_labels(["A", "B", "C"])

set_labels(["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven"])

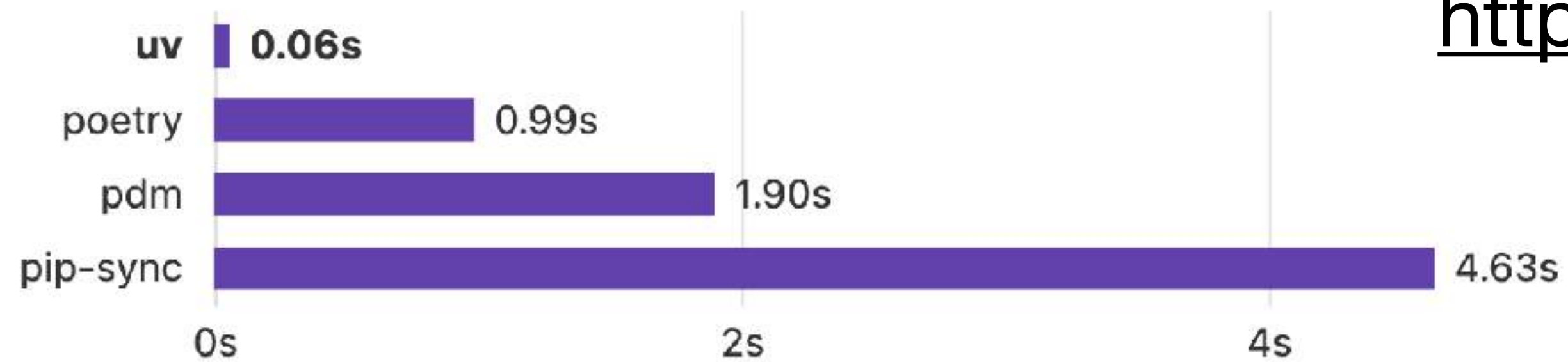
set_labels(["A", "B", "C", "D", "E", "F", "G", "H"])
```





# Package manager UV

An extremely fast Python package and project manager, written in Rust.



<https://docs.astral.sh/uv/>

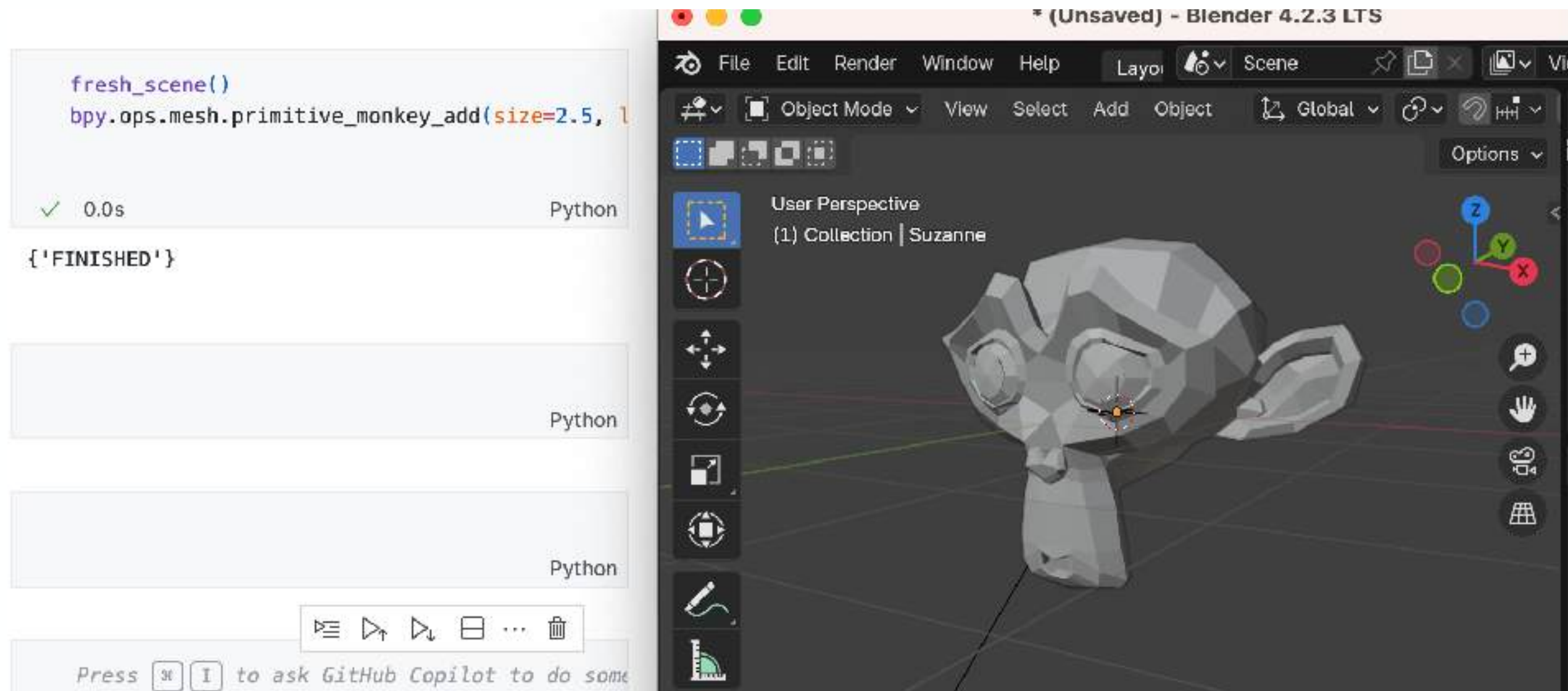
## Install with



```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

# GUI mode

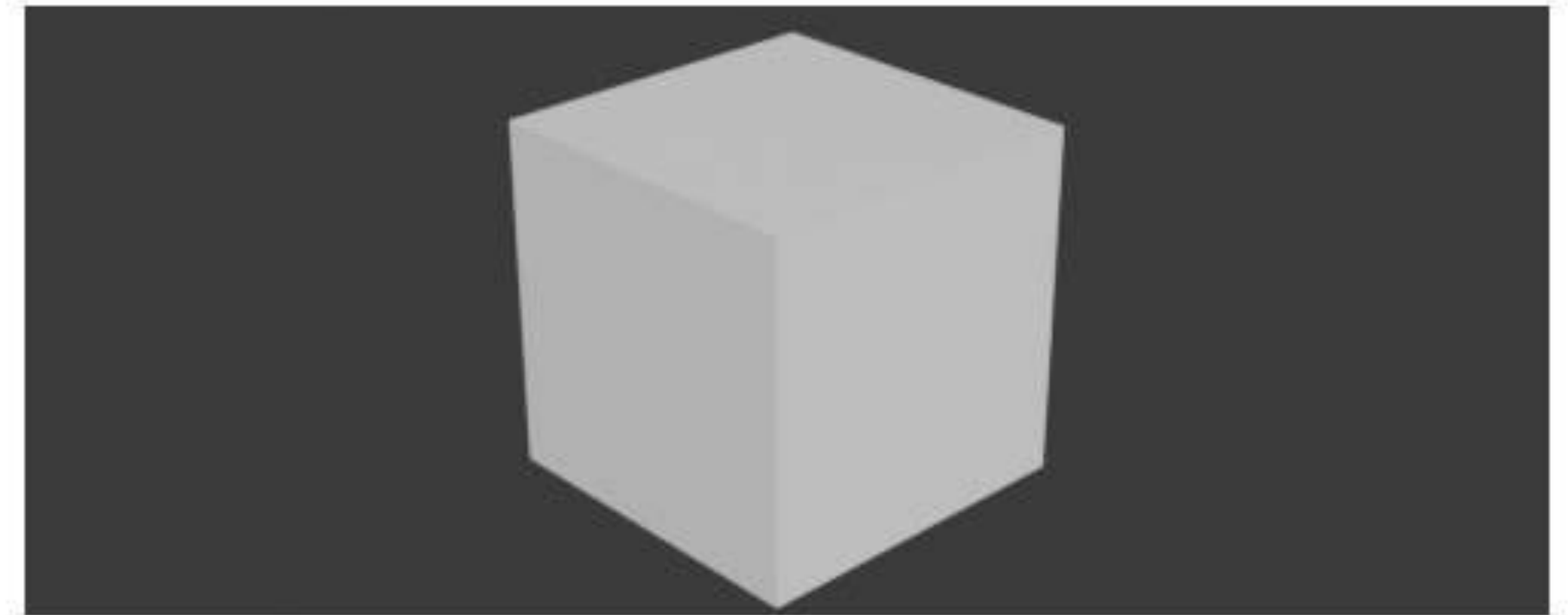
Blender GUI at the side



# Headless mode

Without Blender GUI

```
fresh_scene()
bpy.ops.mesh.primitive_cube_add(size=2, location=(0, 0, 0))
render_result()
```

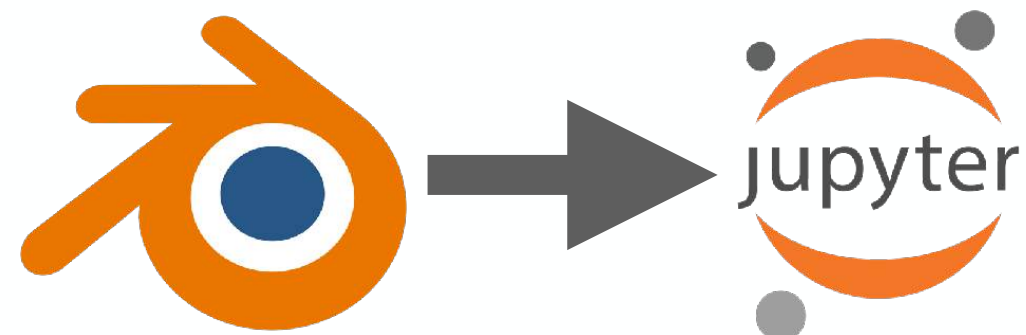


<https://pypi.org/project/bpy/>

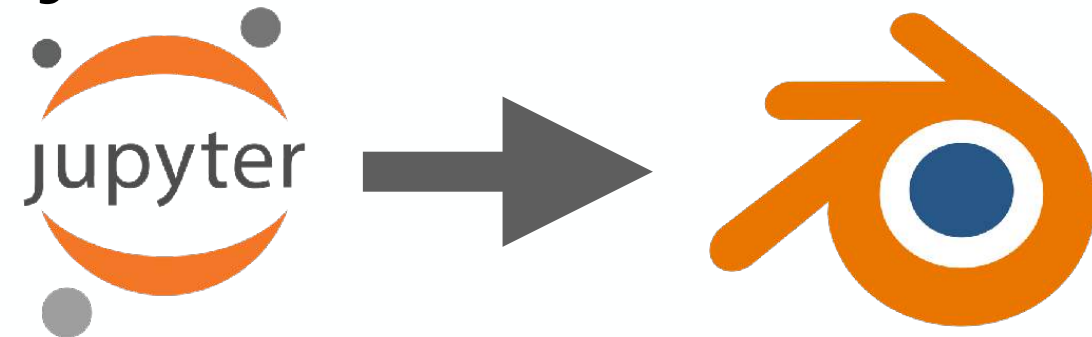
**Two options:**



Blender starts Jupyter



Jupyter starts Blender



[https://github.com/Octoframes/bpy\\_jupyter](https://github.com/Octoframes/bpy_jupyter)

[https://github.com/Octoframes/blender\\_notebook](https://github.com/Octoframes/blender_notebook)

# Headless mode

The screenshot shows the PyPI project page for 'bpy 4.3.0'. The browser address bar shows 'pypi.org/project/bpy/'. The page header includes a search bar with 'Projekte suchen', and navigation links for 'Hilfe', 'Sponsoren', 'Einloggen', and 'Registrieren'. The main content area features the version 'bpy 4.3.0', a 'pip install bpy' button, a 'Neueste Version' badge, and the release date 'Veröffentlicht am: 21. Nov. 2024'. Below this is a subtitle 'Blender as a Python module'. A left sidebar contains a 'Navigation' section with links for 'Projekt-Beschreibung', 'Veröffentlichungs-Historie', and 'Dateien zum Herunterladen', along with a 'Verified details' badge. The main content area has a 'Projekt-Beschreibung' section with the heading 'Blender' and a description of the software and its use as a Python module.

pypi.org/project/bpy/

Projekte suchen

Hilfe Sponsoren Einloggen Registrieren

**bpy 4.3.0**

✓ Neueste Version

Veröffentlicht am: 21. Nov. 2024

pip install bpy

Blender as a Python module

Navigation

- Projekt-Beschreibung
- Veröffentlichungs-Historie
- Dateien zum Herunterladen

Verified details ✓

## Projekt-Beschreibung

### Blender

[Blender](#) is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing.

This package provides Blender as a Python module for use in studio pipelines, web services, scientific research, and more.

# Headless mode

Python Enhancement Proposals | Python » PEP index » PEP 723

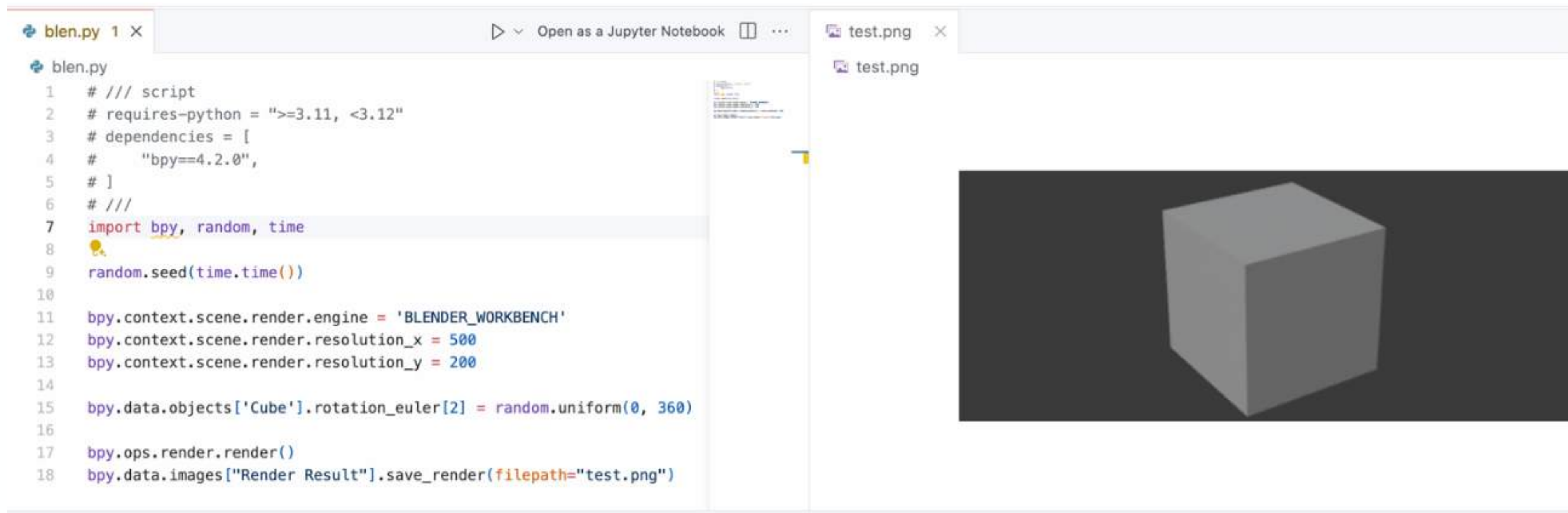
## Contents

- Abstract
- Motivation
- Rationale
- Specification
  - script type
  - Example
- Reference Implementation
- Backwards Compatibility
- Security Implications
- How to Teach This
- Recommendations
- Tooling buy-in
- Rejected Ideas
  - Why not use a comment block resembling requirements.txt?
  - Why not use a multi-line string?
  - Why not reuse core metadata fields?

## PEP 723 – Inline script metadata

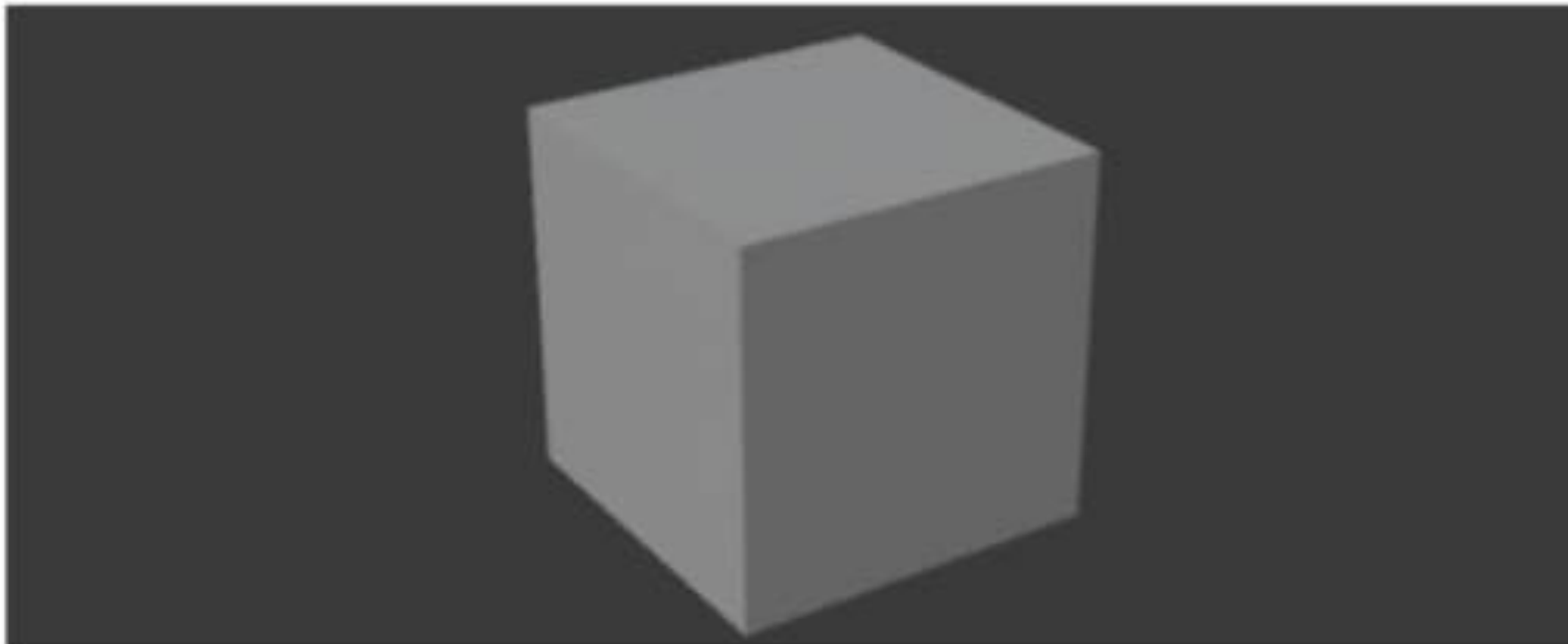
**Author:** Ofek Lev <ofekmeister at gmail.com>  
**Sponsor:** Adam Turner <python at quite.org.uk>  
**PEP-Delegate:** Brett Cannon <brett at python.org>  
**Discussions-To:** [Discourse thread](#)  
**Status:** Final  
**Type:** [Standards Track](#)  
**Topic:** [Packaging](#)  
**Created:** 04-Aug-2023  
**Post-History:** 04-Aug-2023, 06-Aug-2023, 23-Aug-2023, 06-Dec-2023  
**Replaces:** 722  
**Resolution:** 08-Jan-2024

uv run blen.py

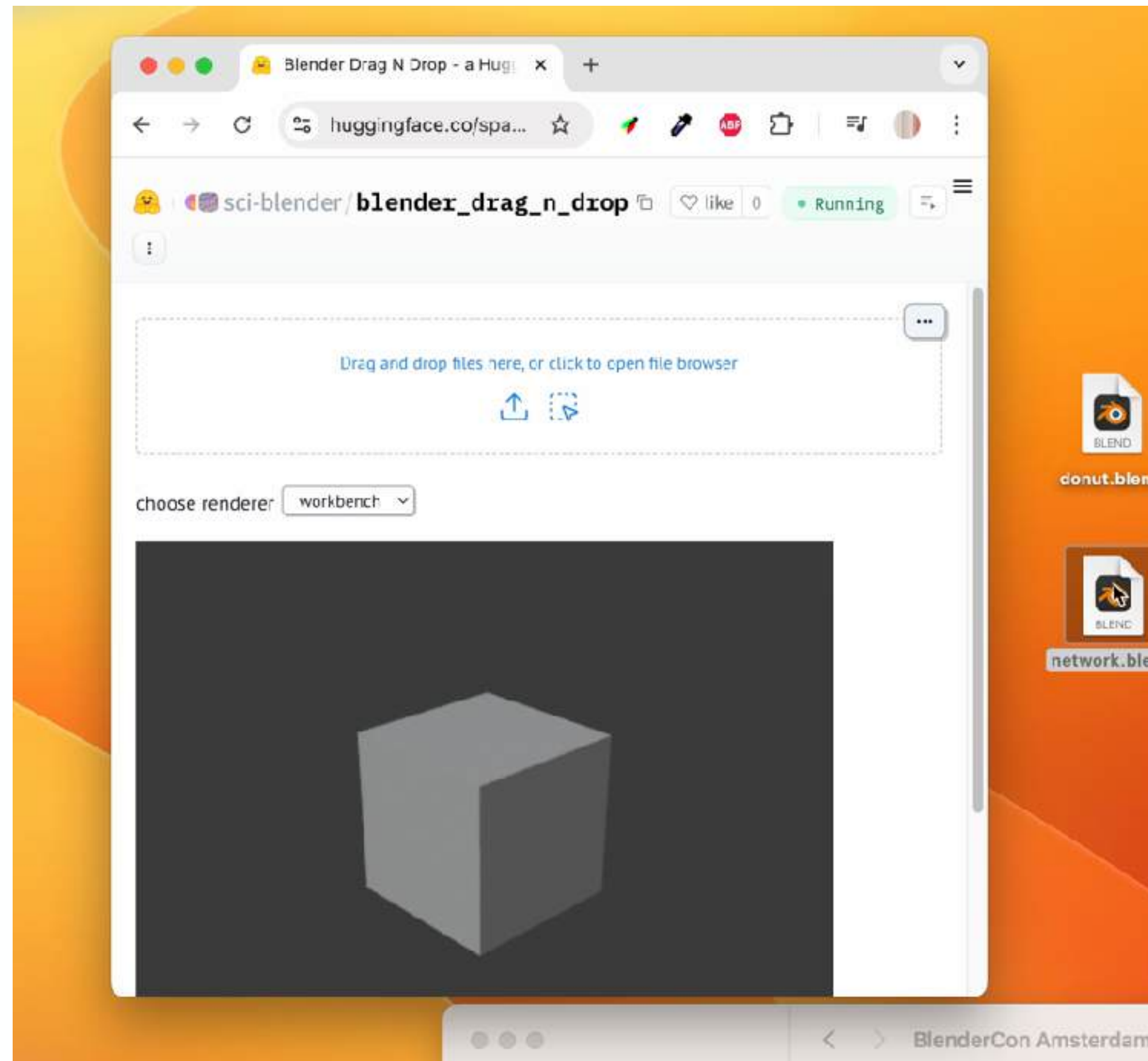


```
blen.py 1 X
blen.py
1 # /// script
2 # requires-python = ">=3.11, <3.12"
3 # dependencies = [
4 #     "bpy==4.2.0",
5 # ]
6 # ///
7 import bpy, random, time
8
9 random.seed(time.time())
10
11 bpy.context.scene.render.engine = 'BLENDER_WORKBENCH'
12 bpy.context.scene.render.resolution_x = 500
13 bpy.context.scene.render.resolution_y = 200
14
15 bpy.data.objects['Cube'].rotation_euler[2] = random.uniform(0, 360)
16
17 bpy.ops.render.render()
18 bpy.data.images["Render Result"].save_render(filepath="test.png")
```

test.png



# Host notebooks online (hugging face)

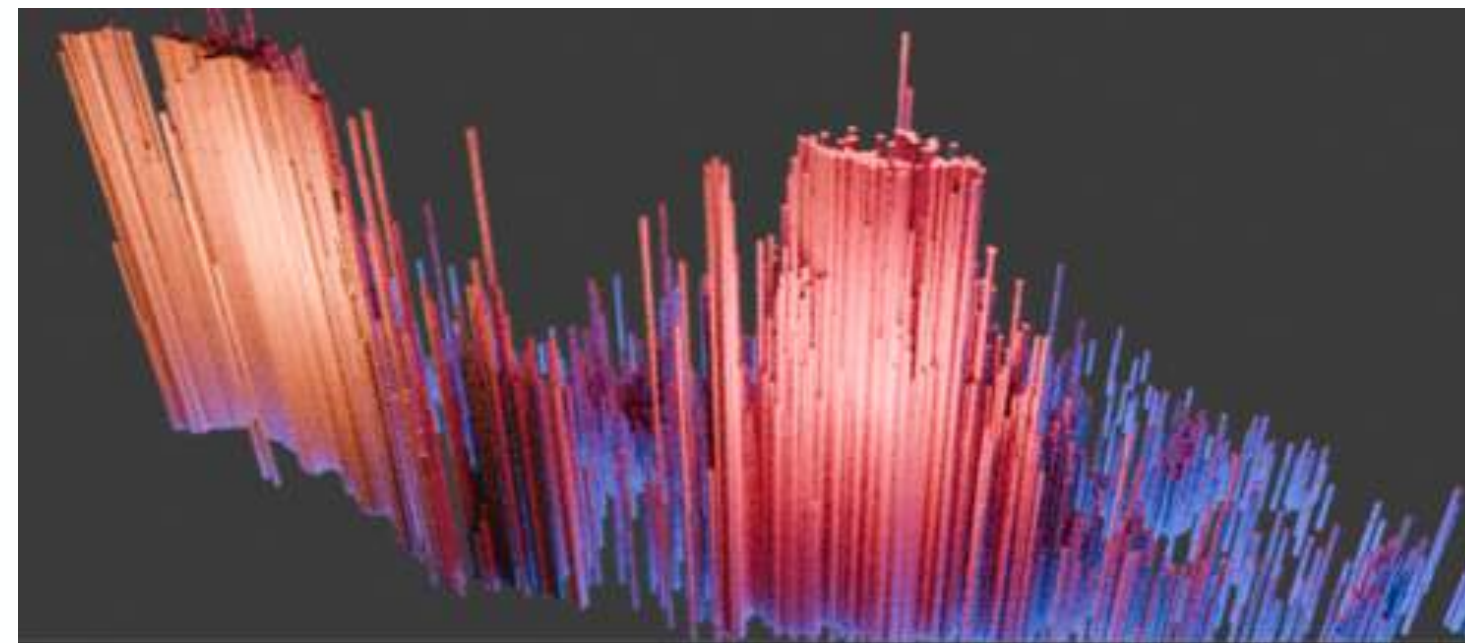


<https://marimo.io/>

[https://huggingface.co/spaces/sci-blender/blender\\_drag\\_n\\_drop](https://huggingface.co/spaces/sci-blender/blender_drag_n_drop)

# Takeaway

- Setup the connection
- Interact with objects
- Data processing pipeline
- Time series
- Links to resources



# Thanks for listening!



- Come join the **Science Blender Discord**: <https://discord.gg/k2Gd2cb4Kk>
- Feel free to reach out after the talk (I'm around till Monday)
- Example Gallery (work in progress): <https://kolibril13.github.io/bpy-gallery/n1objects/>
- Interested in a **one day workshop**?

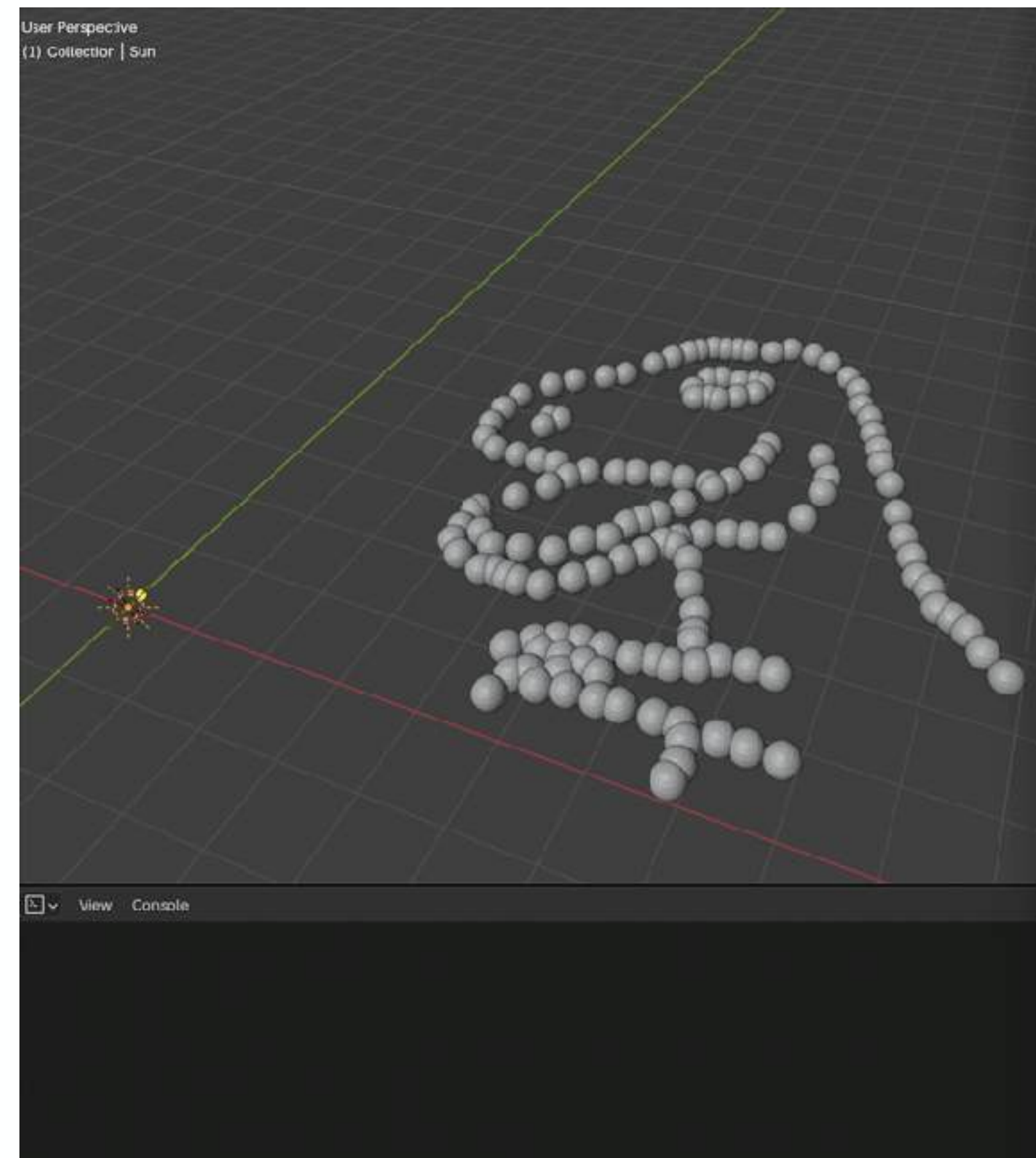


<https://www.linkedin.com/in/jan-hendrik-müller-765014209/>  
<https://bsky.app/profile/kolibril13.bsky.social>


# Backup slides




# Two notebooks at the same time



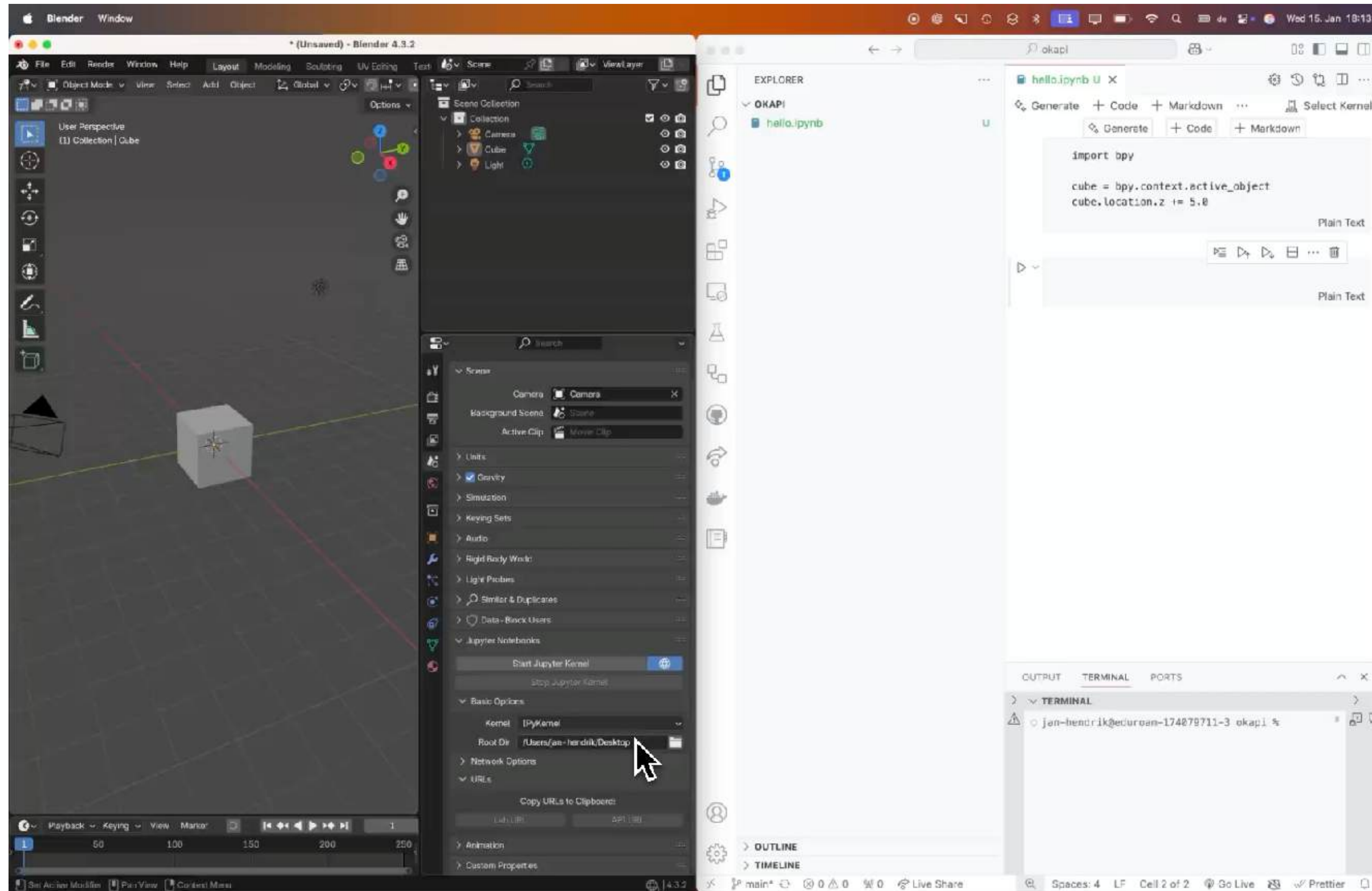
```
n2data_simple.ipynb Python 3 (ipykernel)
[100]: fresh_scene()
# Create a new mesh and object for the point cloud
mesh = bpy.data.meshes.new("HelloDenoMesh")
my_point_obj = bpy.data.objects.new("HelloDeno", mesh)
points = [mathutils.Vector((x, y, 0)) for x, y in zip(x_values1, y_values1)] # Create points
# Apply the points to the mesh
mesh.from_pydata(vertices=points, edges=[], faces=[])
mesh.update()
bpy.context.collection.objects.link(my_point_obj)
modifier = my_point_obj.modifiers.new(name="GeometryNodes", type='NODES')
modifier.node_group = node_group_place_spheres
my_point_obj.modifiers["GeometryNodes"]["Socket_2"] = 0.2 # Set radius
camera_from_above()
render_result()
[123]: my_point_obj.modifiers["GeometryNodes"]["Socket_2"] = 0.1 # Set radius
render_result()
```



```
n2data_simple-ref.ipynb Python 3 (ipykernel)
[179]: fresh_scene()
# Create a new mesh and object for the point cloud
mesh = bpy.data.meshes.new("HelloDenoMesh")
my_point_obj = bpy.data.objects.new("HelloDeno", mesh)
points = [mathutils.Vector((x, y, 0)) for x, y in zip(x_values, y_values)] # Create points f
# Apply the points to the mesh
mesh.from_pydata(vertices=points, edges=[], faces=[])
mesh.update()
bpy.context.collection.objects.link(my_point_obj)
modifier = my_point_obj.modifiers.new(name="GeometryNodes", type='NODES')
modifier.node_group = node_group_place_spheres
my_point_obj.modifiers["GeometryNodes"]["Socket_2"] = 0.2 # Set radius
camera_from_above()
render_result()
[123]: my_point_obj.modifiers["GeometryNodes"]["Socket_2"] = 0.1 # Set radius
render_result()
```

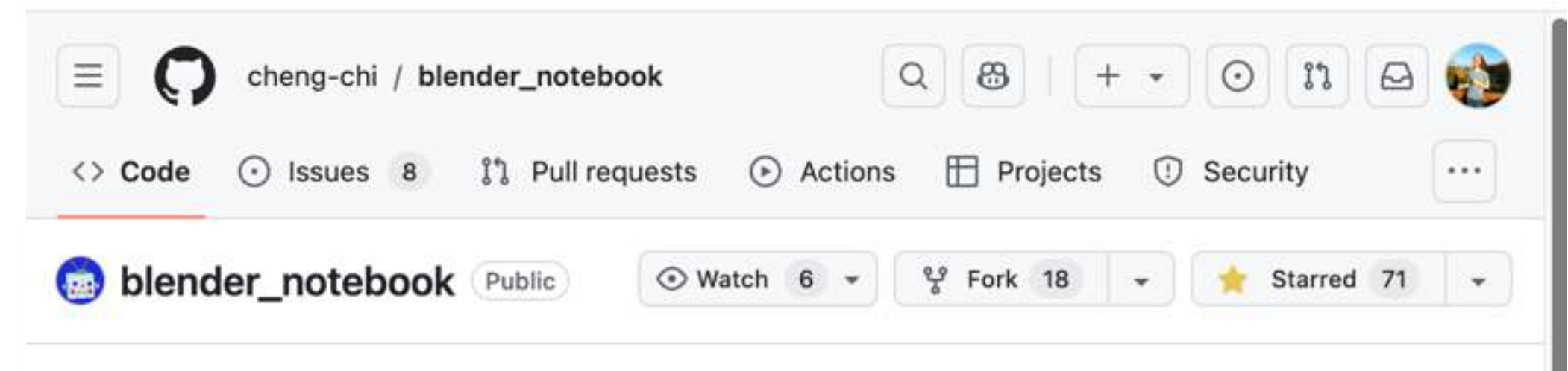


# Connecting to VS Code



# Install GUI-mode

```
uvx blender_notebook install --blender-exec="/Applications/Blender.app/Contents/MacOS/Blender"  
  
uvx --python 3.11 jupyter lab  
  
# ... and select Kernel
```



Full instructions at

[https://kolibril13.github.io/bpy-gallery/n0getting\\_started/](https://kolibril13.github.io/bpy-gallery/n0getting_started/)