Blender for Computer Scientists:

Basics,
Scripting,
Rendering Figures

More resources

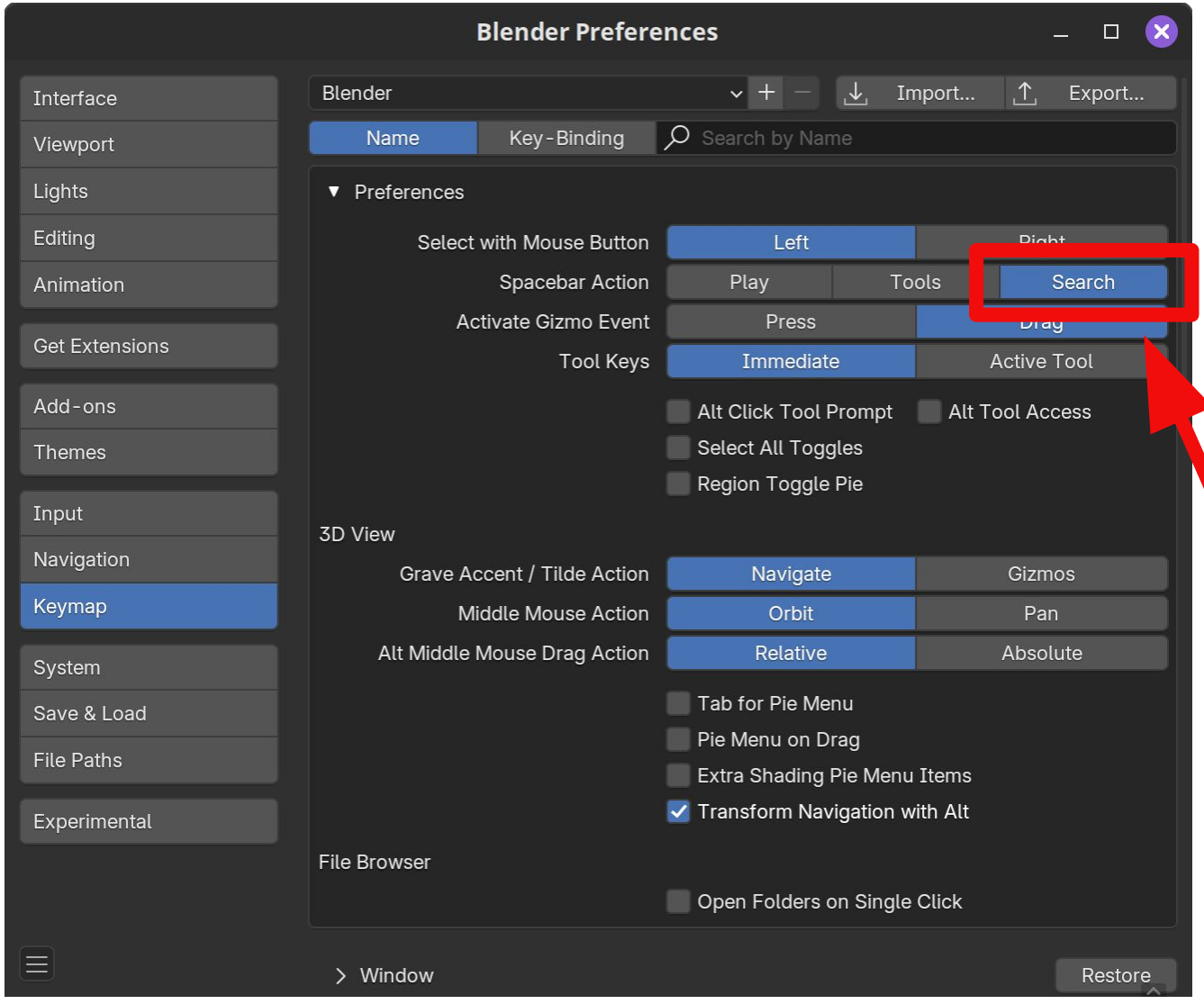
- Silvia’s blog entries:
  - https://www.silviasellan.com/posts/blender_figure/
- BlenderToolbox
  - https://github.com/HTDerekLiu/BlenderToolbox
- BlenderGuru
  - https://www.youtube.com/@blenderguru
  - https://www.blenderguru.com/articles/cycles-shader-encyclopedia

General Blender disclaimers

- Be careful with tutorials made before 2.80 (July 2019)
- I’m currently using Blender 4.2.3 (LTS)
- Blender 5.0 will release in November 2025

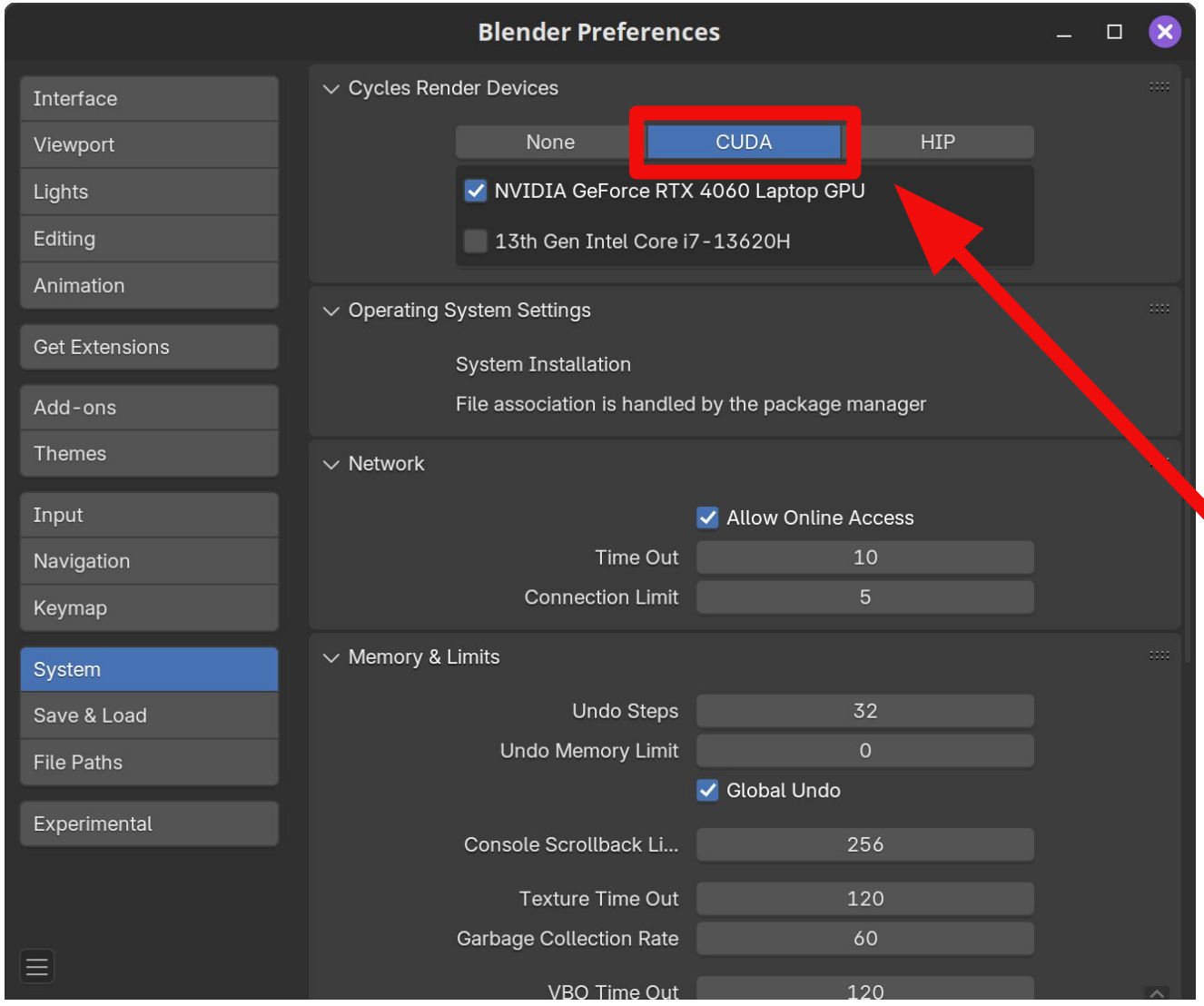
- It’s better to have a Numpad, but not mandatory

Before we begin…
Some preferences to edit



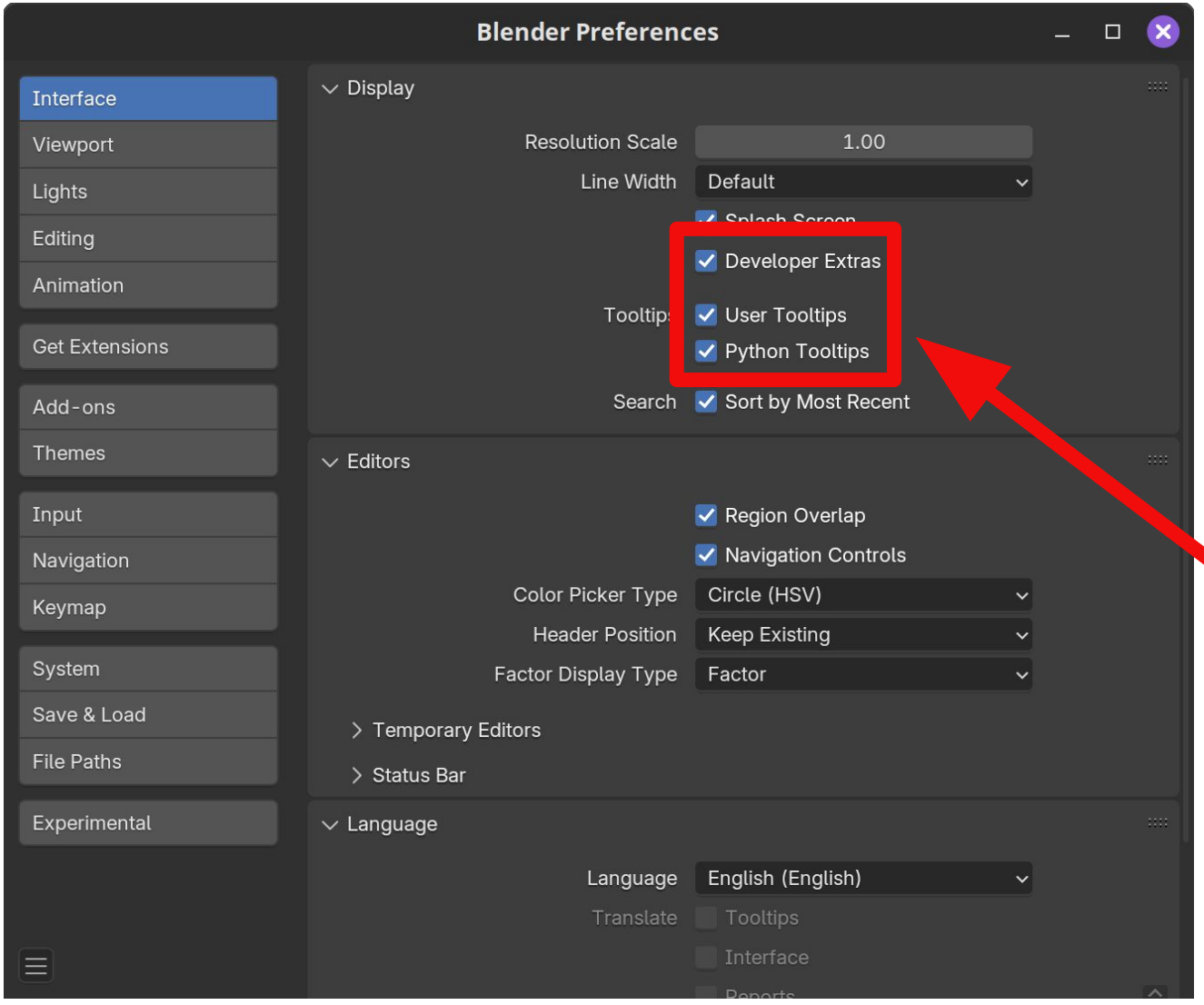
Keymap menu

No more need to remember where everything is!



System menu

Wait less for renders! Use your GPU

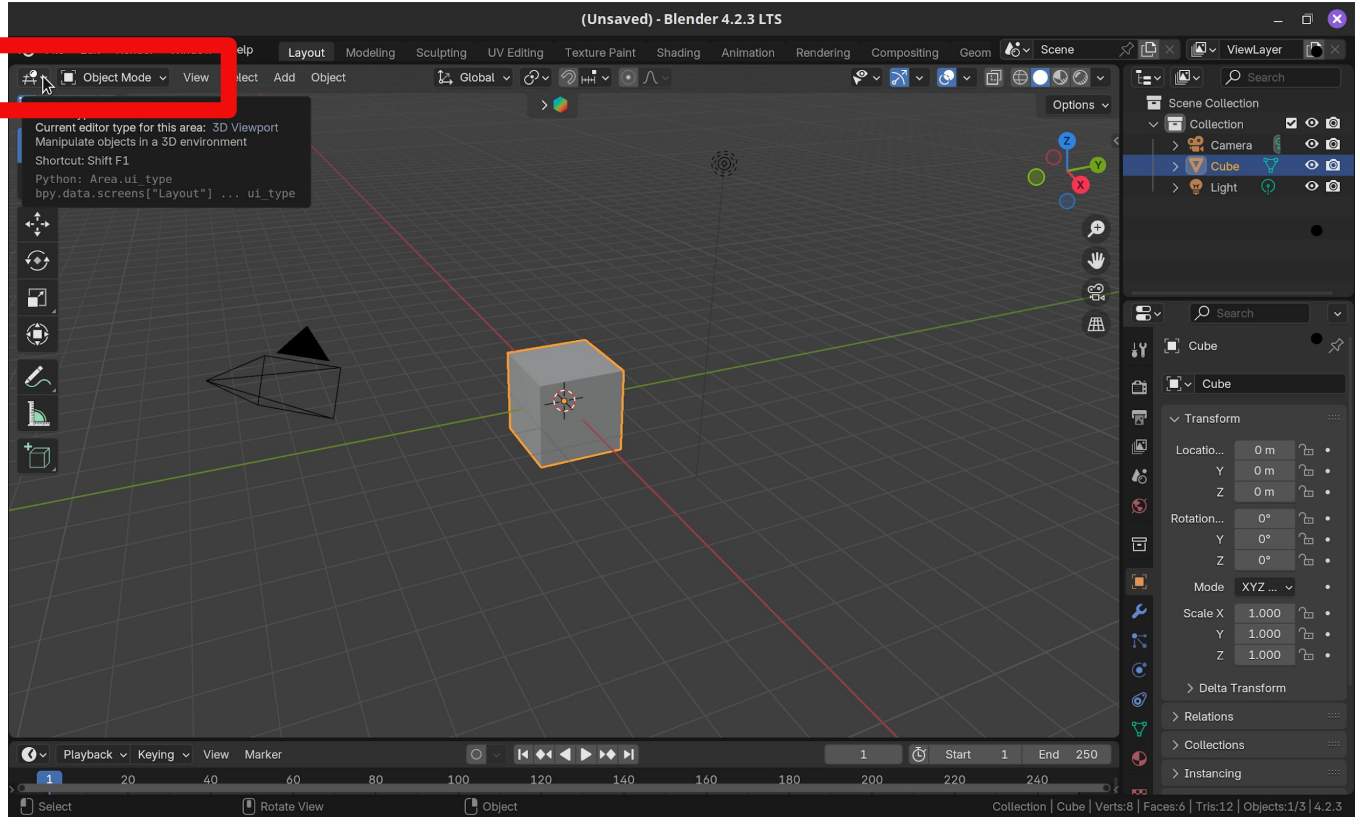


Interface menu

More information!
Learn how to perform actions with code instead of UI!

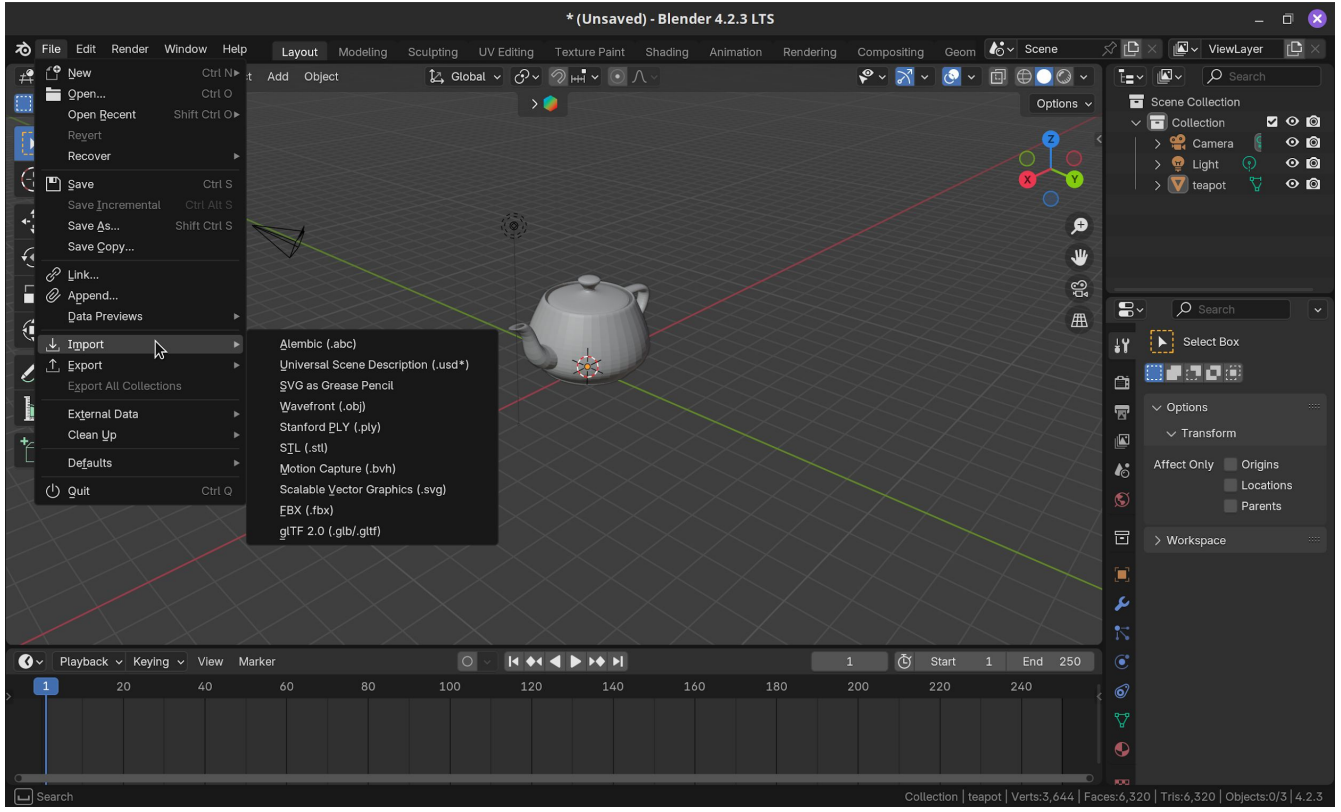
Basics

The 3D Viewport



Main place for interactions.

Default interaction type is Object mode.



Import objects in one of the supported formats (drag-and-drop also an option)

Render by hitting F12

Transforms in Object mode

- G for position
- R for rotation
- S for scale
- Hit x, y, z keys to then lock transforms to an axis
- Or type numbers to enter exact coordinate (tab to change dimensions)

Transforms in Object mode

- , to modify orientation of transform (local / global, etc)
- . to modify origin of transform (individual origin, cursor, median)

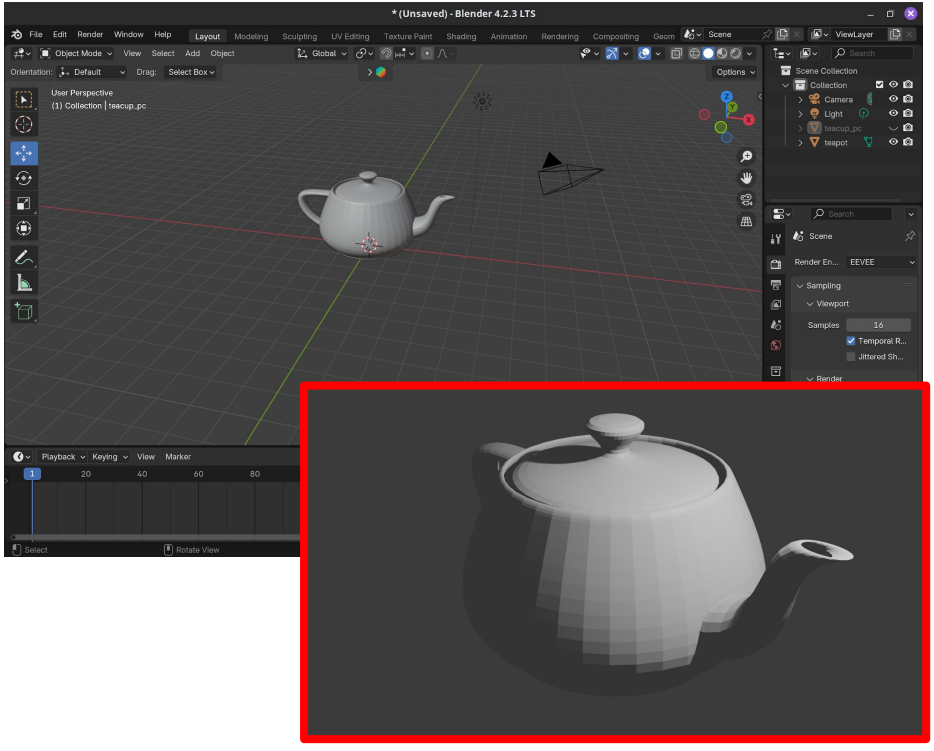
Setting up the camera

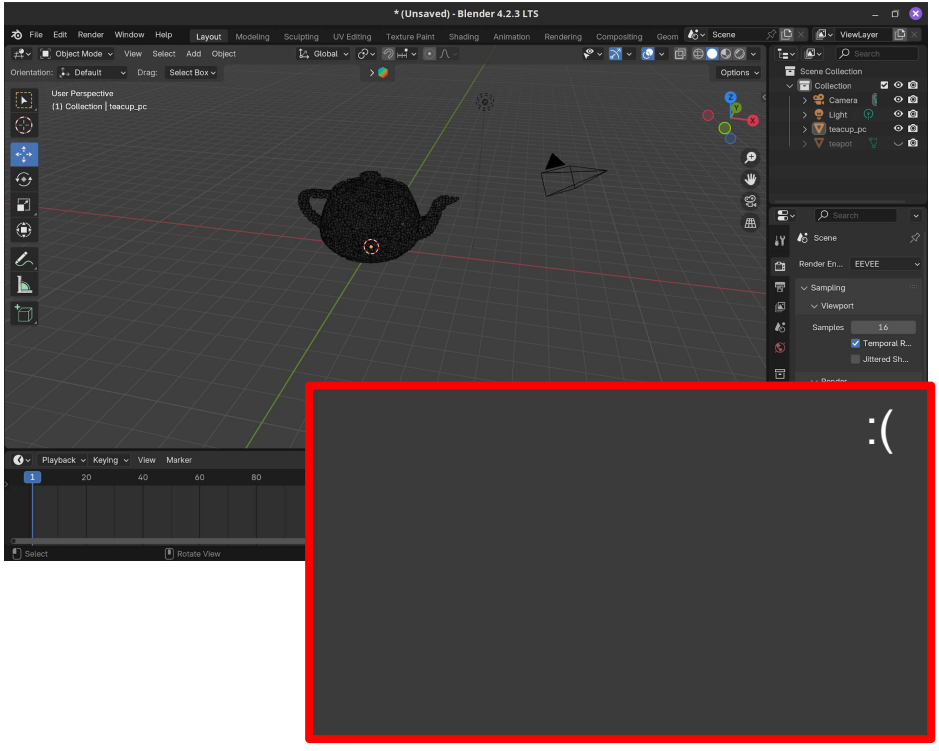
- Move it around, like an object
- Or: align your view however you like, then use “align camera to view”

Modifications in Edit mode

- C to select faces like a brush
- H to hide selected faces, Alt + H to unhide
  - The hidden faces will stay hidden in renders!
- E to extrude, I to inset

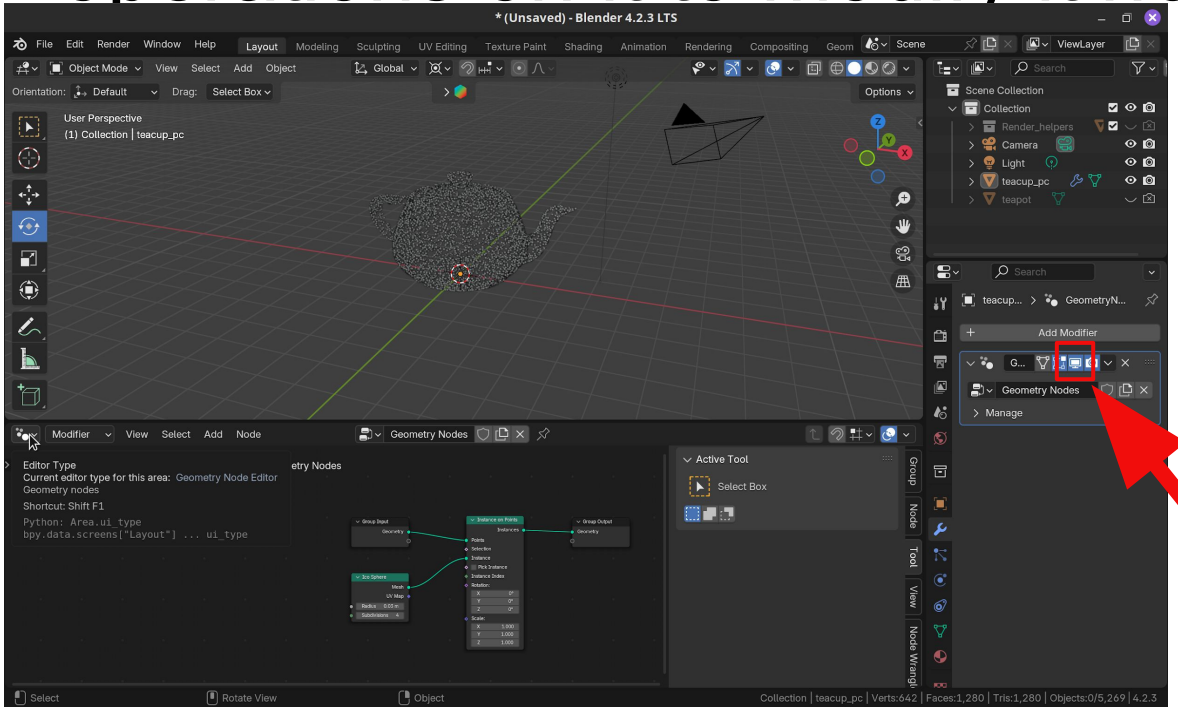
Warning! Point clouds cannot be rendered by default

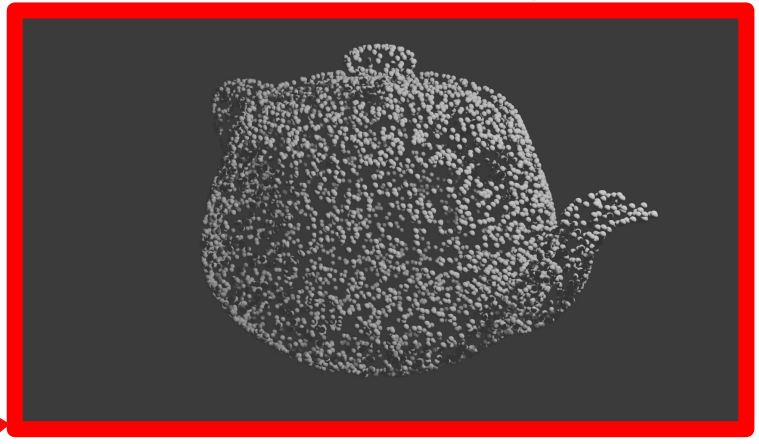




Geometry Nodes

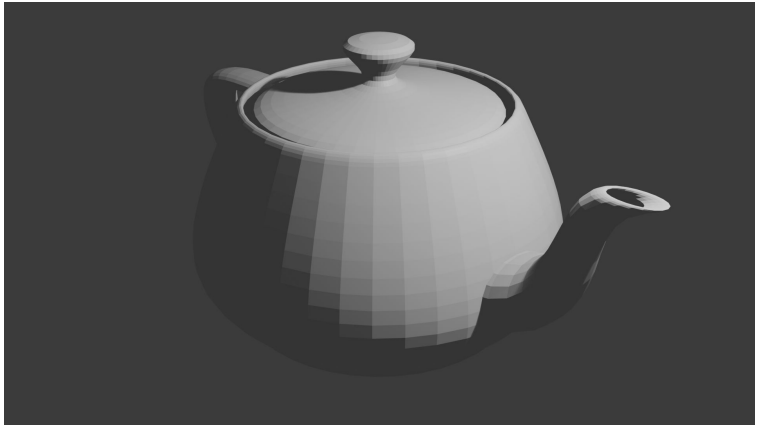
- Given some object, perform a series of operations on it to modify it *non-destructively*



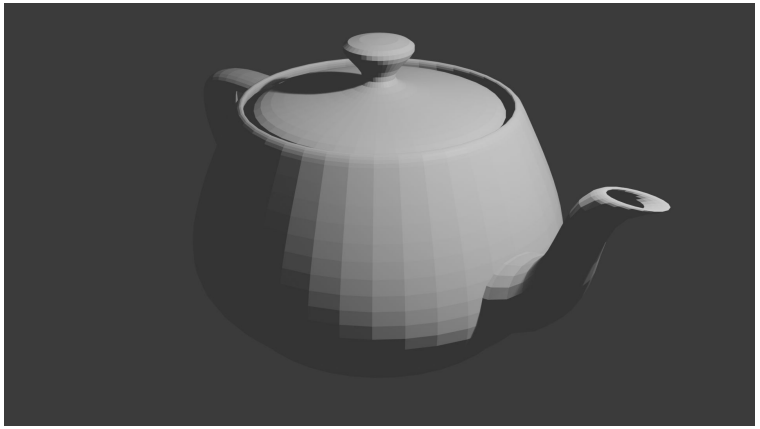


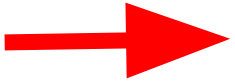
Disable the node in the viewport if you notice significant lag

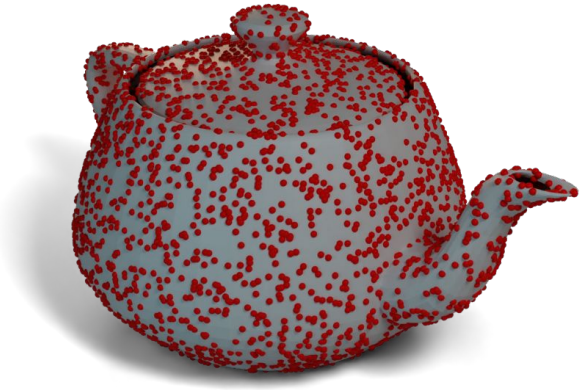
How to make images look good?



How to make images look good?

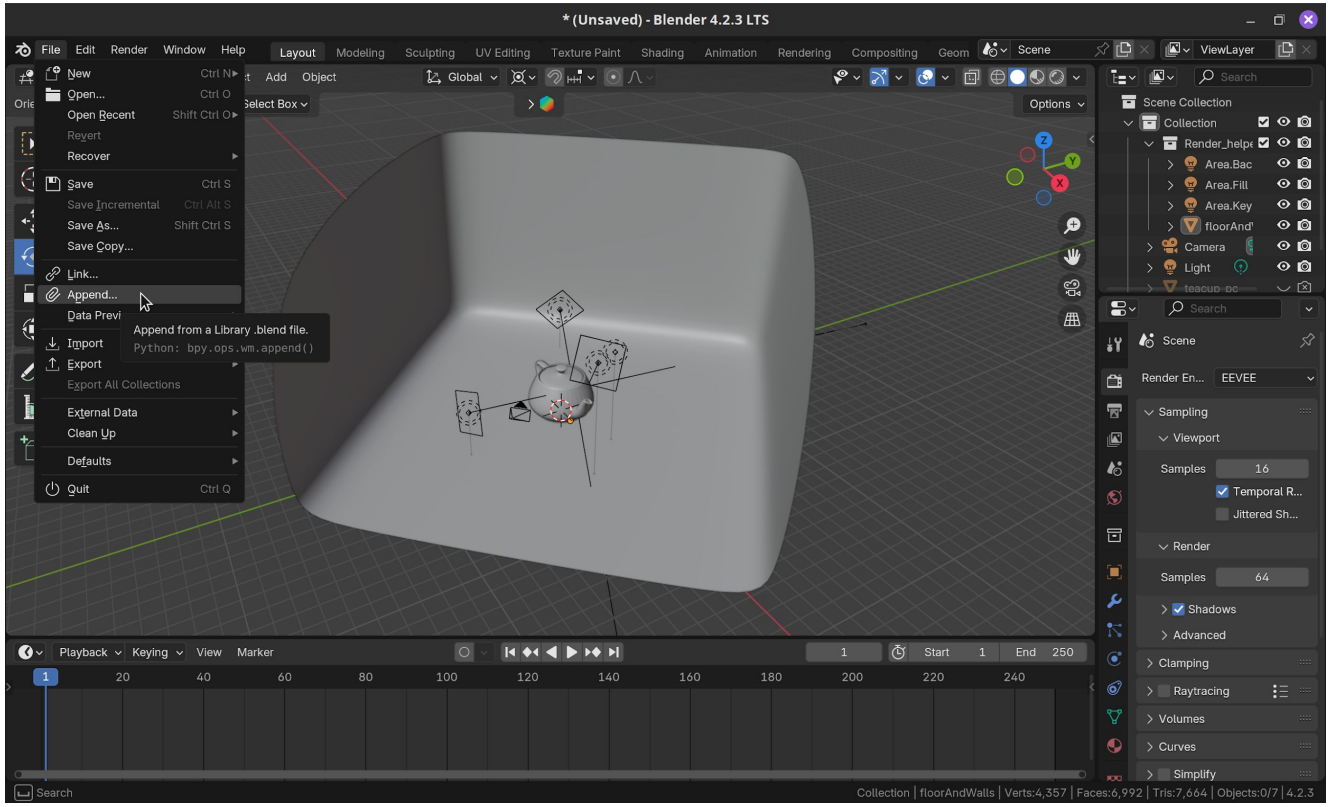






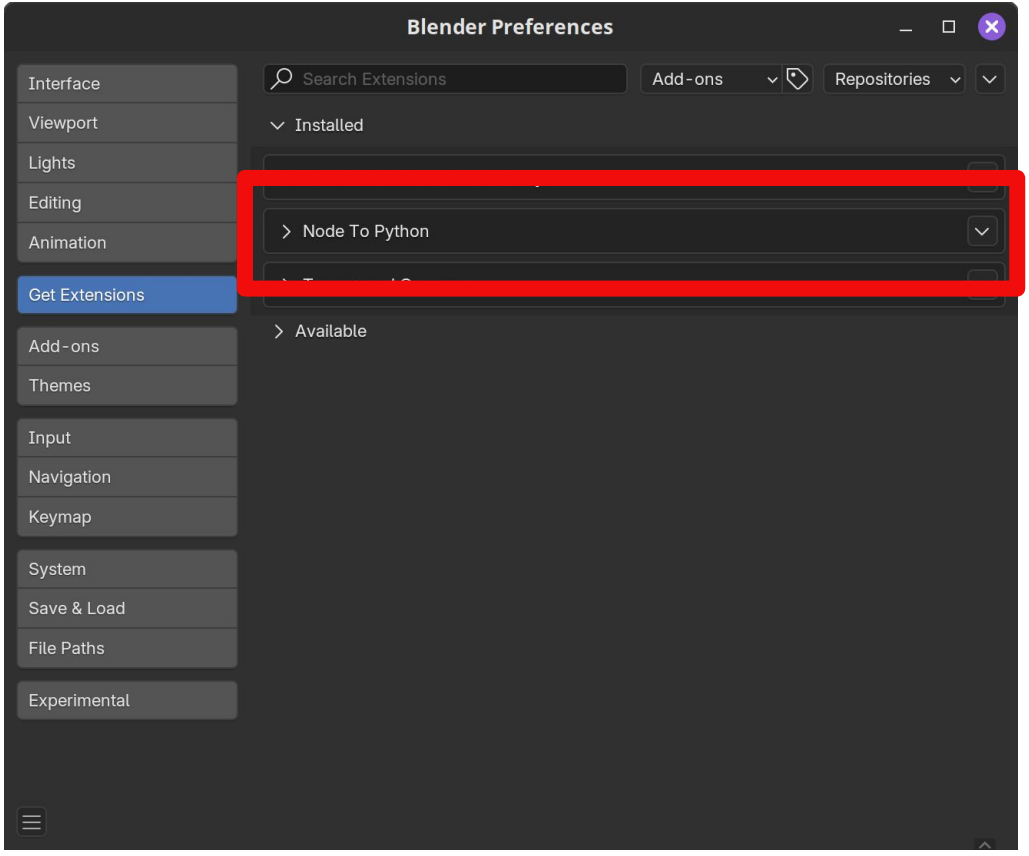
How to make something once and then never worry about it again?

Appending other .blend files



Create some setup in blender (eg. three-point lighting), then reuse it in all your other files

Node to python extension



Code for point cloud rendering:

```
import bpy, mathutils

def geometry_nodes_node_group():
    geometry_nodes = bpy.data.node_groups.new(type = 'GeometryNodeTree', name = "Geometry Nodes")
    geometry_nodes.color_tag = 'NONE'
    geometry_nodes.description = ""


    geometry_nodes.is_modifier = True

    #geometry_nodes interface
    #Socket Geometry
    geometry_socket = geometry_nodes.interface.new_socket(name = "Geometry", in_out='OUTPUT', socket_type = 'NodeSocketGeometry')
    geometry_socket.attribute_domain = 'POINT'

    #Socket Geometry
    geometry_socket_1 = geometry_nodes.interface.new_socket(name = "Geometry", in_out='INPUT', socket_type = 'NodeSocketGeometry')
    geometry_socket_1.attribute_domain = 'POINT'


    #initialize geometry_nodes nodes
    #node Group Input
    group_input = geometry_nodes.nodes.new("NodeGroupInput")
    group_input.name = "Group Input"

    #node Group Output
    group_output = geometry_nodes.nodes.new("NodeGroupOutput")
    group_output.name = "Group Output"
    group_output.is_active_output = True

    #node Ico Sphere
    ico_sphere = geometry_nodes.nodes.new("GeometryNodeMeshIcoSphere")
    ico_sphere.name = "Ico Sphere"
    #Radius
    ico_sphere.inputs[0].default_value = 0.029999999329447746
    #Subdivisions
    ico_sphere.inputs[1].default_value = 4

    #node Instance on Points
    instance_on_points = geometry_nodes.nodes.new("GeometryNodeInstanceOnPoints")
    instance_on_points.name = "Instance on Points"
    #Selection
    instance_on_points.inputs[1].default_value = True
    #Pick Instance
    instance_on_points.inputs[3].default_value = False
    #Instance Index
    instance_on_points.inputs[4].default_value = 0
    #Rotation
    instance_on_points.inputs[5].default_value = (0.0, 0.0, 0.0)
    #Scale
    instance_on_points.inputs[6].default_value = (1.0, 1.0, 1.0)



    #Set locations
    group_input.location = (-340.0, 0.0)
    group_output.location = (200.0, 0.0)
    ico_sphere.location = (-340.0126037597656, -135.51634216308594)
    instance_on_points.location = (-59.0997314453125, 4.5728759765625)

    #Set dimensions
    group_input.width, group_input.height = 140.0, 100.0
    group_output.width, group_output.height = 140.0, 100.0
    ico_sphere.width, ico_sphere.height = 140.0, 100.0
    instance_on_points.width, instance_on_points.height = 140.0, 100.0

    #initialize geometry_nodes links
    #instance_on_points.Instances -> group_output.Geometry
    geometry_nodes.links.new(instance_on_points.outputs[0], group_output.inputs[0])
    #group_input.Geometry -> instance_on_points.Points
    geometry_nodes.links.new(group_input.outputs[0], instance_on_points.inputs[0])
    #ico_sphere.Mesh -> instance_on_points.Instance
    geometry_nodes.links.new(ico_sphere.outputs[0], instance_on_points.inputs[2])
    return geometry_nodes

geometry_nodes = geometry_nodes_node_group()
```

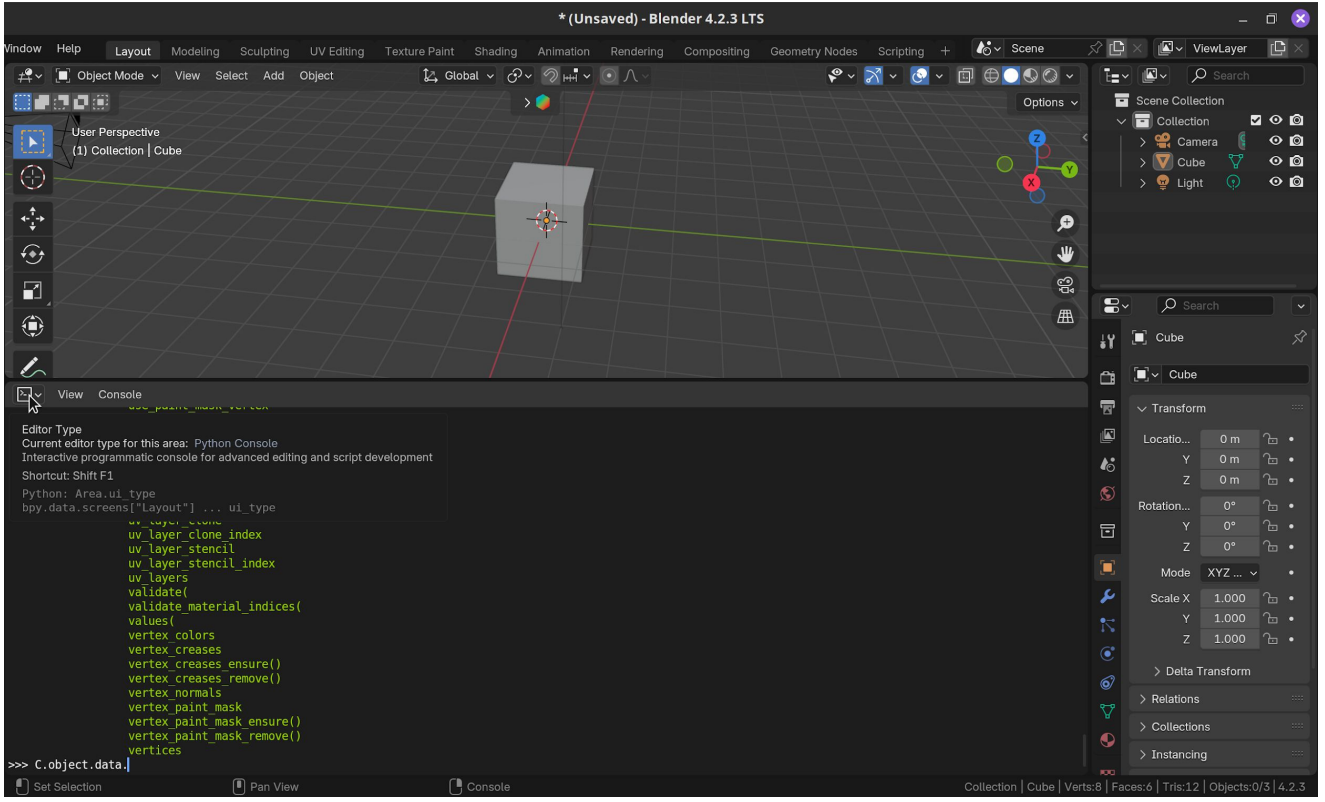
Scripting in Blender

Scripting in Blender

- Blender comes with its own installation of Python

- NOTE:
  - Blender 4.0 - uses python 3.10
  - Blender 4.1 + uses python 3.11

The Python Console



To prototype and learn the API

Press tab to see the available attributes / functions

NOTE: You can paste scripts in here too, but beware of spaces!

The bpy module

- Provides access to data inside Blender

- If you want to do math in Blender, you need to import mathutils (Vectors, Quaternions, etc)

Accessing things

- bpy.data: Get a reference to something by name or by index
  - ex: bpy.data.objects[‘Cube’] → gets the default cube (if it still exists!)
- bpy.context: Get a reference to whatever is currently relevant
  - ex: bpy.context.object → gets the active object
  - You’ll likely mainly use this for collection, scene, view_layer

Accessing things

- object.data contains most of the things we care about:
  - .polygons : list of faces
  - .vertices : list of vertices
    - .co : location of the vertex, *in local space*
  - .materials : materials assigned to object
- object.matrix_world : to bring object to world space

Performing actions with bpy.ops

- NOTE: anything you do will affect the selected / active object
- Anything starting with bpy.ops will not return useful information

```
# x will contain STATUS code returned by the operator
x = bpy.ops.mesh.primitive_plane_add(size=100)
# the plane will actually be the current active object
plane = bpy.context.object
```

Performing actions with bpy.ops

- NOTE: anything you do will affect the selected / active object
- Anything starting with bpy.ops will not return useful information

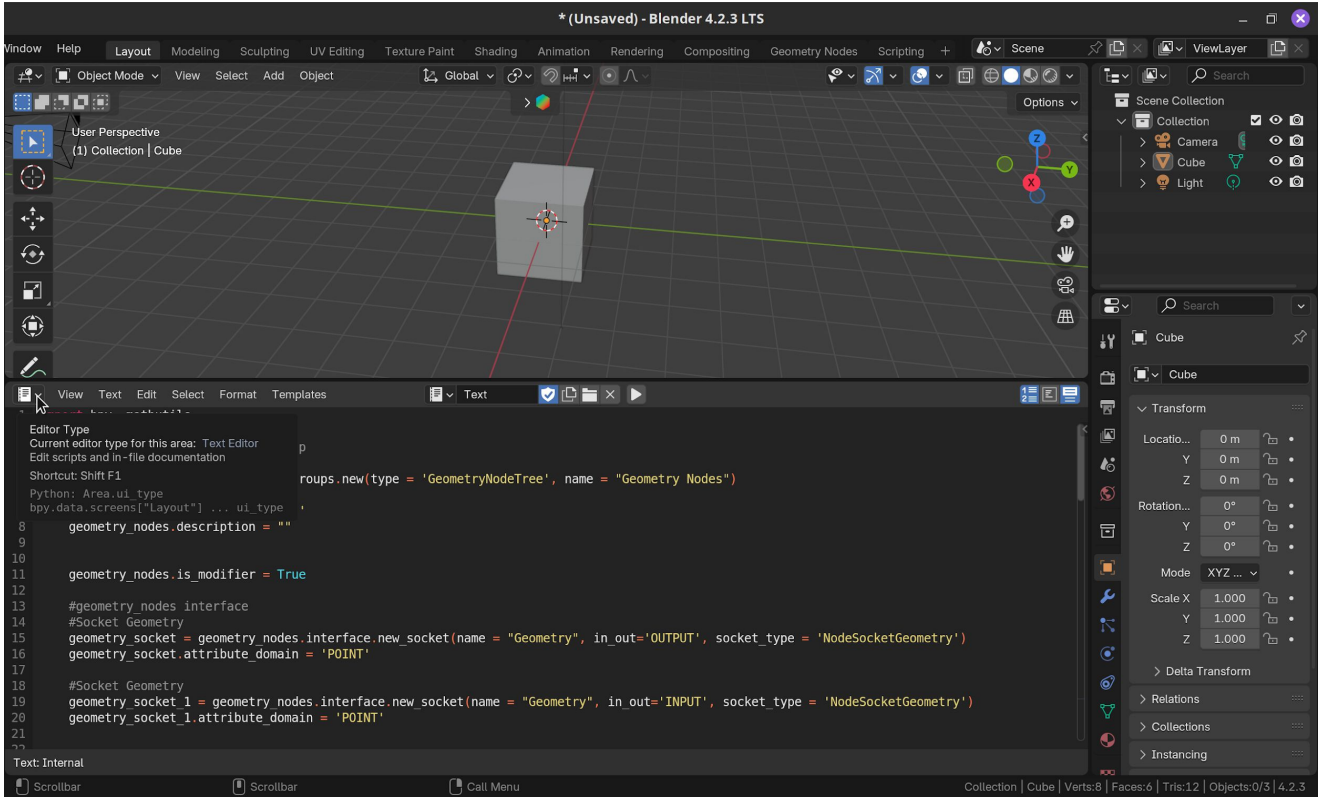
```
# object must be SELECTED for operator to affect them. All
selected objects will be affected!
cube = bpy.data.objects[‘Cube’]
bpy.ops.object.select_all(action='DESELECT')
cube.select_set(True)
bpy.ops.object.origin_set(type="ORIGIN_CURSOR")
```

Performing actions with bpy.data

- bpy.data will *actually* return useful things

```
# red will hold the new material
red = bpy.data.materials.new("Red")
red.use_nodes = True
red.node_tree.nodes['Principled BSDF'].inputs['Base
Color'].default_value = (1, 0, 0, 1)
bpy.data.objects[‘Cube’].data.materials.append(red)
```

The Python Editor



To run medium-scale scripts

NOTE: print statements do not show up anywhere

The Command Line

- blender --background --python /path/to/script.py

- No limits on what you can do, BUT there are a couple of tricks to know first

Import local module

- Assume you have code in file secondary.py
- You will run blender -b -P main.py
- Add to main.py:

```
import bpy, os, sys

blend_dir = os.path.dirname(bpy.data.filepath)
if blend_dir not in sys.path:
    sys.path.append(blend_dir)
```

Use numpy/scipy/etc in Blender

- Navigate to wherever Blender is stored
  - Windows: "C:\Program Files\Blender Foundation\Blender [Version]\[Version]\python\bin"
  - Write in terminal:

```
./python -m pip install numpy scipy whatever
```

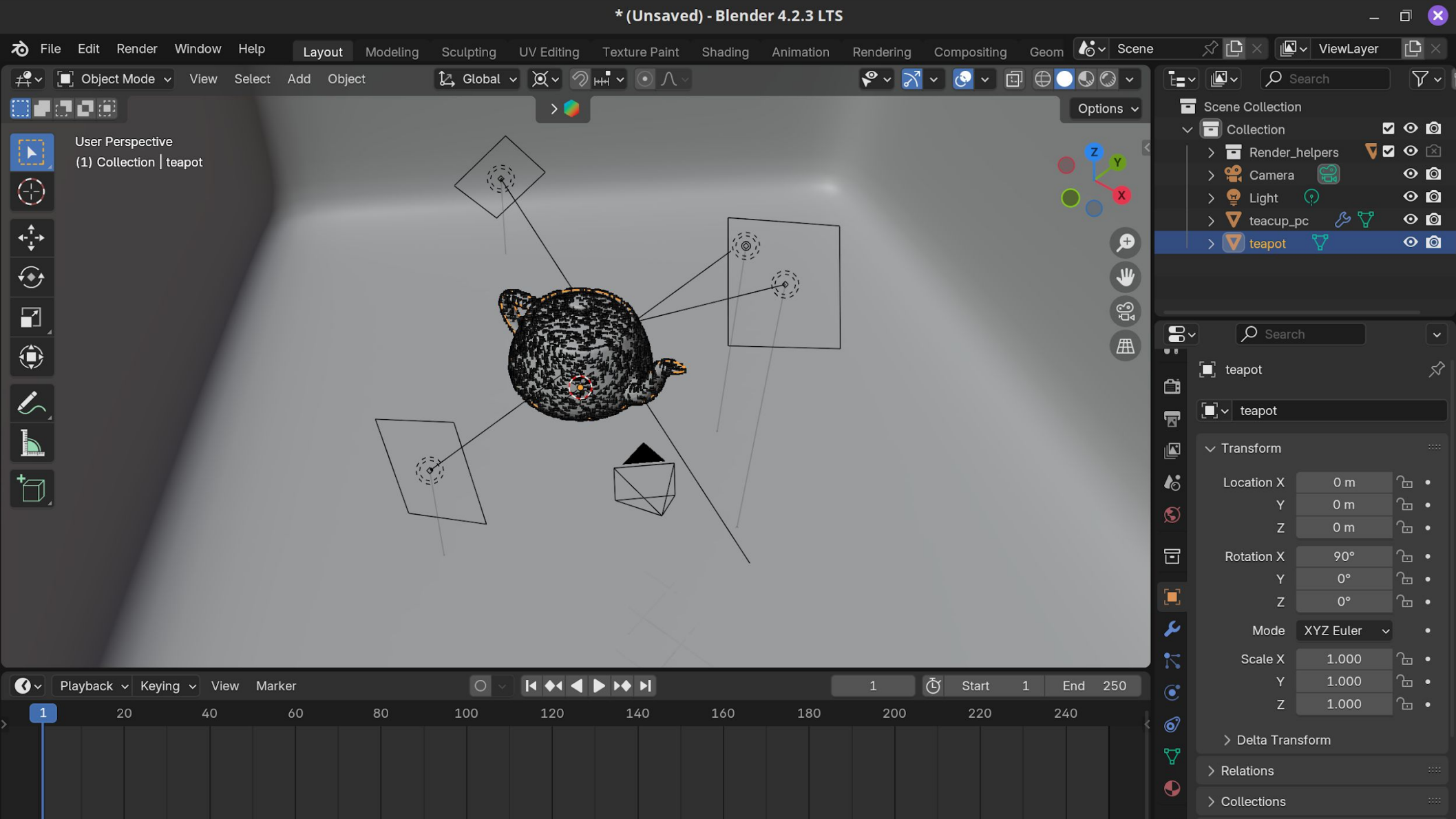
Pass arguments to your script

- Assume you care about the width of your image
- Write in main.py :

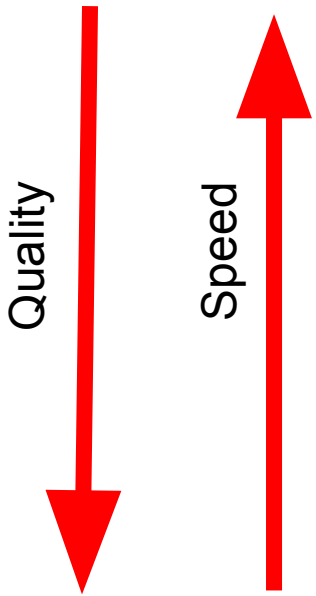
```
parser = argparse.ArgumentParser(description='My
main.')
parser.add_argument('--width', type=int,
default=1080,help='width')
argv = sys.argv[sys.argv.index("--") + 1:]
args = parser.parse_args(argv)
```

- Then call script using: blender -b -P main.py -- --width 700

Rendering Figures


* (Unsaved) - Blender 4.2.3 LTS
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geom
Scene
ViewLayer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | teapot
Search
Scene Collection
Collection
Render_helpers
Camera
Light
teacup_pc
teapot
Search
teapot
teapot
Transform
Location X 0 m
Y 0 m
Z 0 m
Rotation X 90°
Y 0°
Z 0°
Mode XYZ Euler
Scale X 1.000
Y 1.000
Z 1.000
Delta Transform
Relations
Collections
Playback Keying View Marker
1
Start 1
End 250
1 20 40 60 80 100 120 140 160 180 200 220 240

3 rendering options



- Workbench:
  - Output exactly what you see in Blender's default view
- EEVEE:
  - Rasterization engine
- Cycles:
  - Ray-tracing engine

Shadow catchers (Cycles only)

- Whatever you use as a floor should not be showing up in your render. BUT, it would be nice to still have the floor shadows

```
bpy.data.objects["floor"].is_shadow_catcher =
```

I always dump this in my file:

Activate cycles, make film transparent, etc.

```
context = bpy.context
scene = bpy.data.scenes["Scene"]
render = scene.render


render.engine = 'CYCLES'  # ('BLENDER_EEVEE', 'BLENDER_WORKBENCH',
'CYCLES')
render.image_settings.color_mode = 'RGBA'  # ('RGB', 'RGBA', ...)
render.image_settings.color_depth = '8'  # ('8', '16')
render.image_settings.file_format = 'PNG'  # ('PNG', 'OPEN_EXR',
'JPEG, ...)
render.image_settings.compression = 50
render.resolution_x = 1080
render.resolution_y = 1080
render.film_transparent = True
render.use_persistent_data = True


scene.display_settings.display_device = "sRGB"
scene.view_settings.view_transform = "Filmic"
scene.view_settings.look = "Very High Contrast"



scene.cycles.preview_samples = 32
scene.cycles.samples = 100
scene.cycles.use_denoising = True
scene.cycles.device = 'GPU'
```

Set up some basic post-processing to reduce noisiness

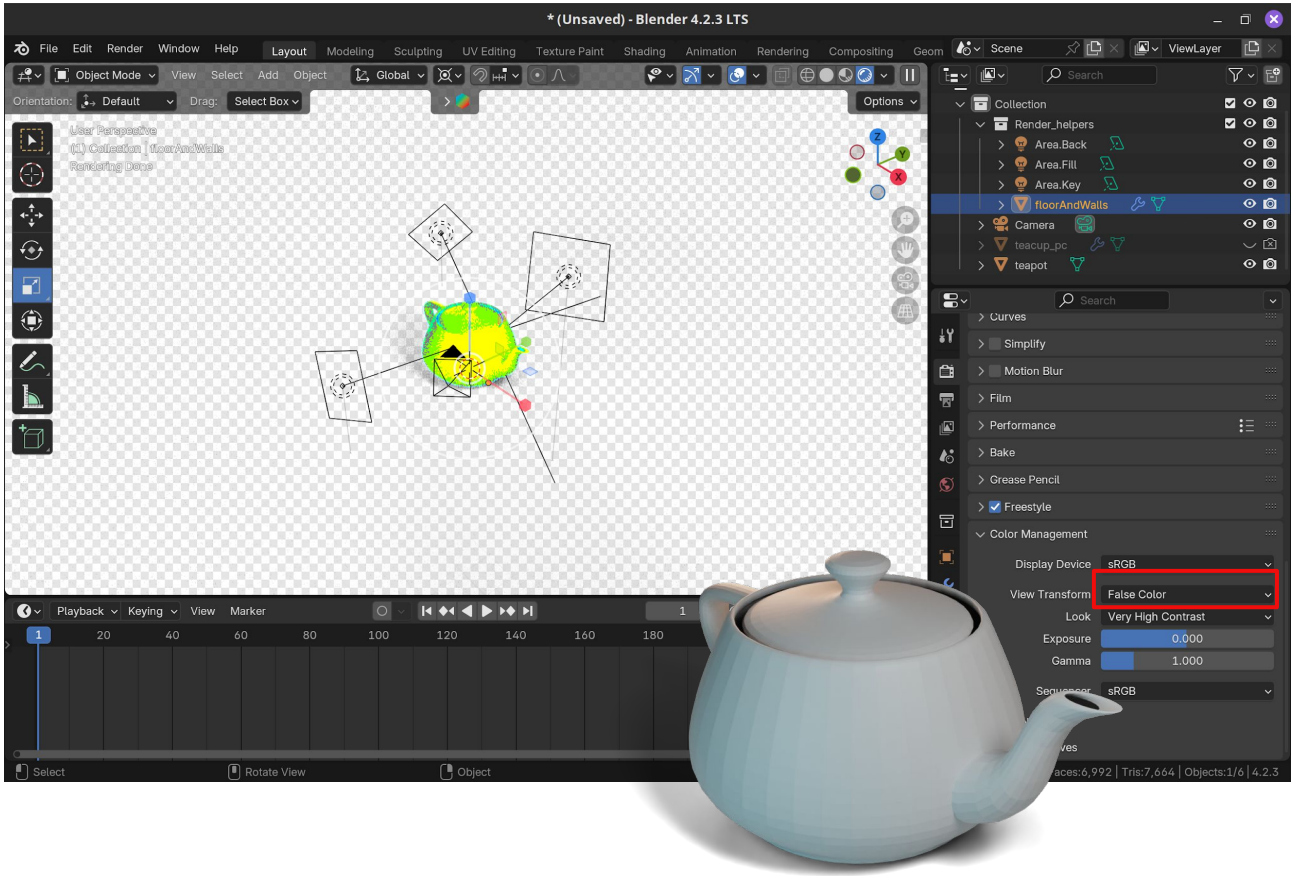
```
bpy.context.scene.use_nodes = True
ntree = bpy.context.scene.node_tree
#out = ntree.nodes.new("CompositorNodeComposite")
out = ntree.nodes['Composite']
out.location = (850, 40)
innn = ntree.nodes['Render Layers']
ntree.links.new(innn.outputs['Image'],
out.inputs['Image'])


ramp = ntree.nodes.new("CompositorNodeValToRGB")
ramp.location = (350, -100)
ramp_s = ramp.color_ramp.elements[0]
ramp_e = ramp.color_ramp.elements[1]
ramp_s.position = 0.1
ramp_s.color = (0, 0, 0, 0)
ramp_e.color = (0, 0, 0, 1)
ntree.links.new(innn.outputs['Alpha'],
ramp.inputs['Fac'])
ntree.links.new(ramp.outputs['Alpha'],
out.inputs['Alpha'])
bright = ntree.nodes.new("CompositorNodeBrightContrast")
bright.location = (350, 130)
bright.use_premultiply = False
```

Why is post-processing important?

vs

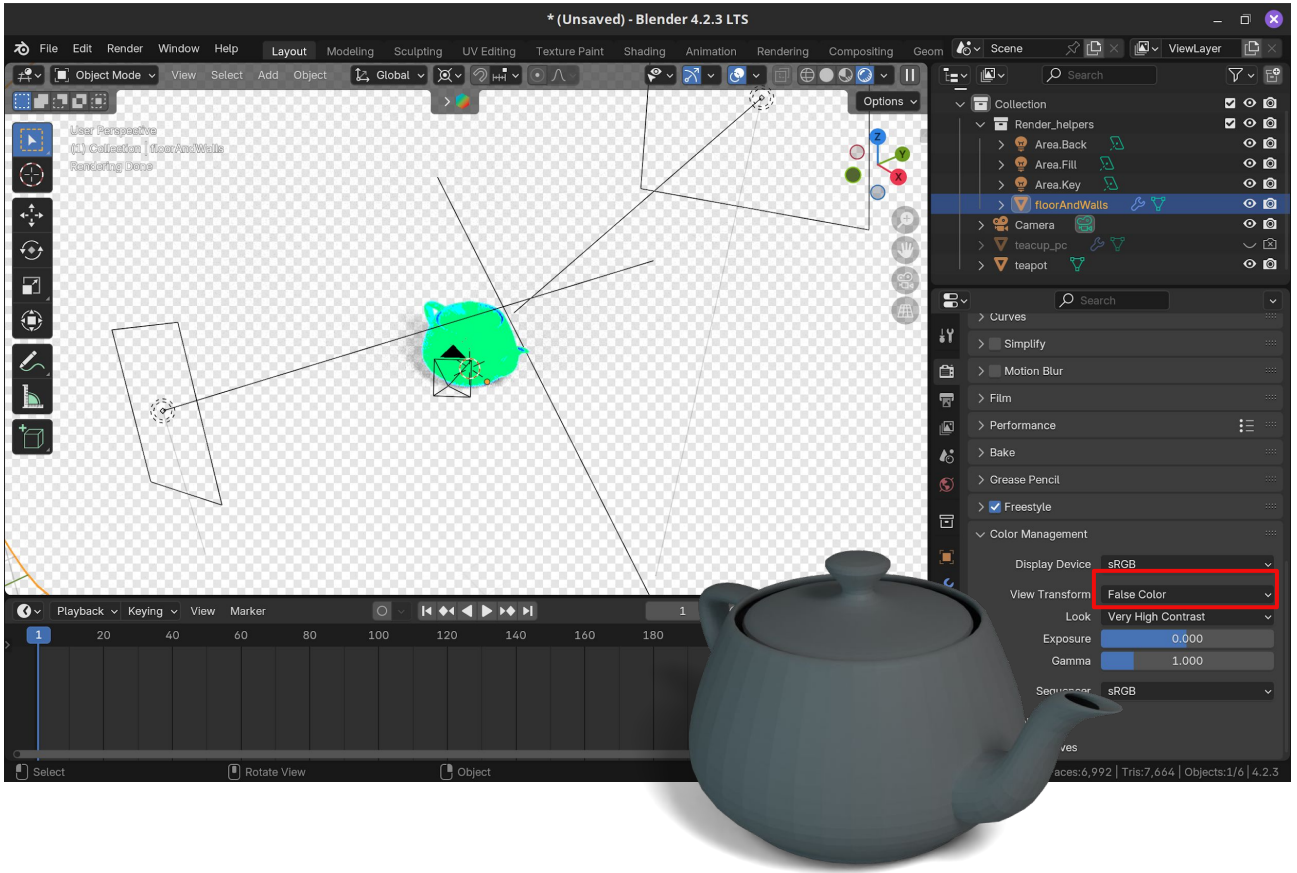
How to place light at right distance



Render (camera icon)
→ Color management
→ View transform
→ False color

Yellow means too bright,

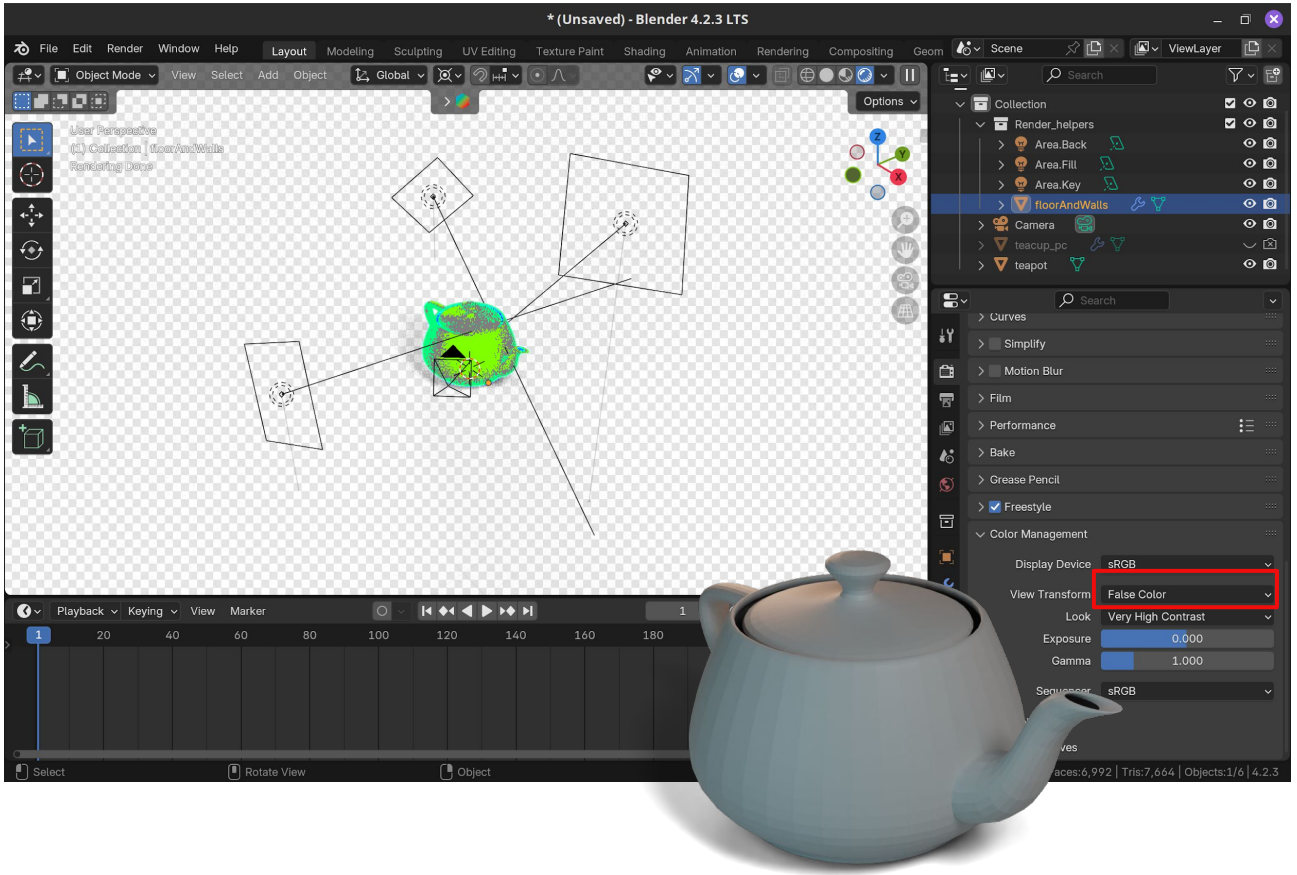
How to place light at right distance



Render (camera icon)
→ Color management
→ View transform
→ False color

Yellow means too bright, dark green means too dark,

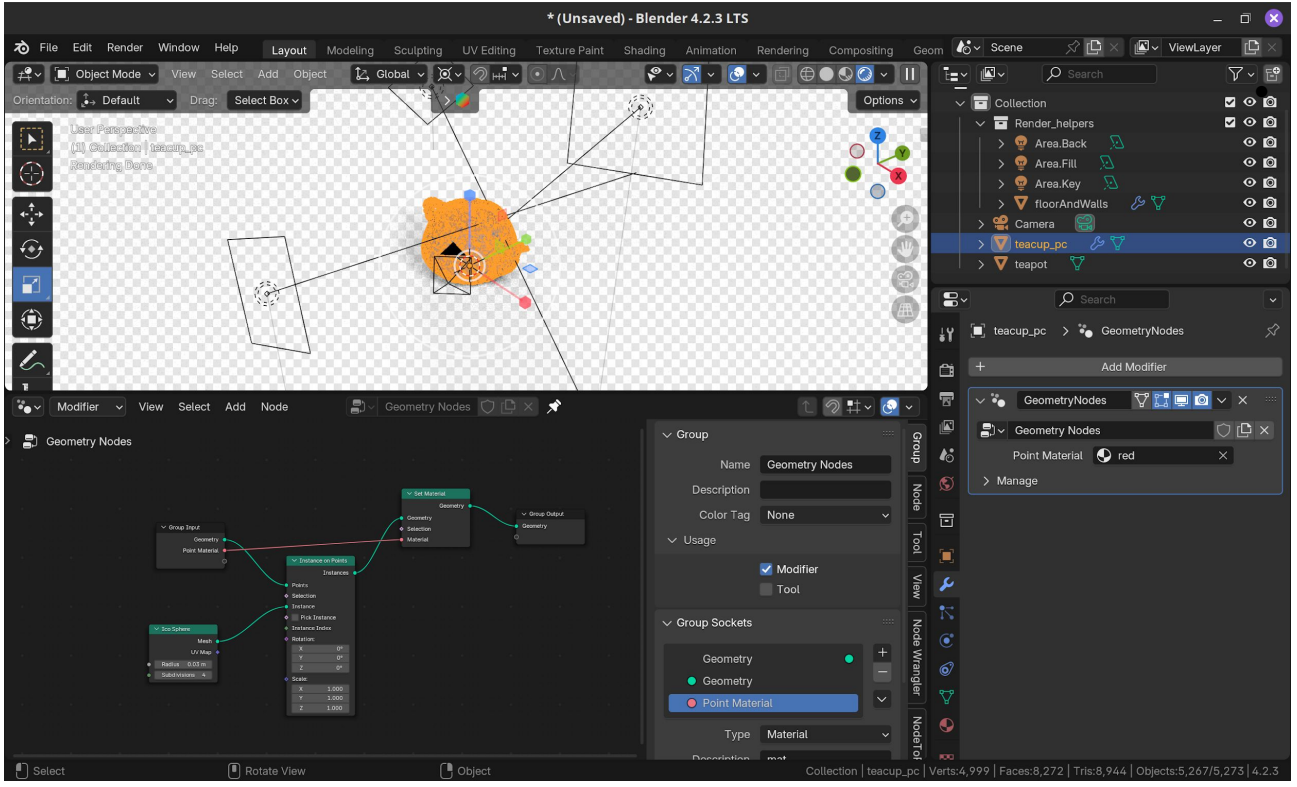
How to place light at right distance



Render (camera icon)
→ Color management
→ View transform
→ False color

Yellow means too bright, dark green means too dark, grey means ok

Setting a material for the point cloud



One more addition to geonodes: add a new input socket, which will add a custom material to the point cloud (don’t make one geonode per point cloud!)

Thank you!