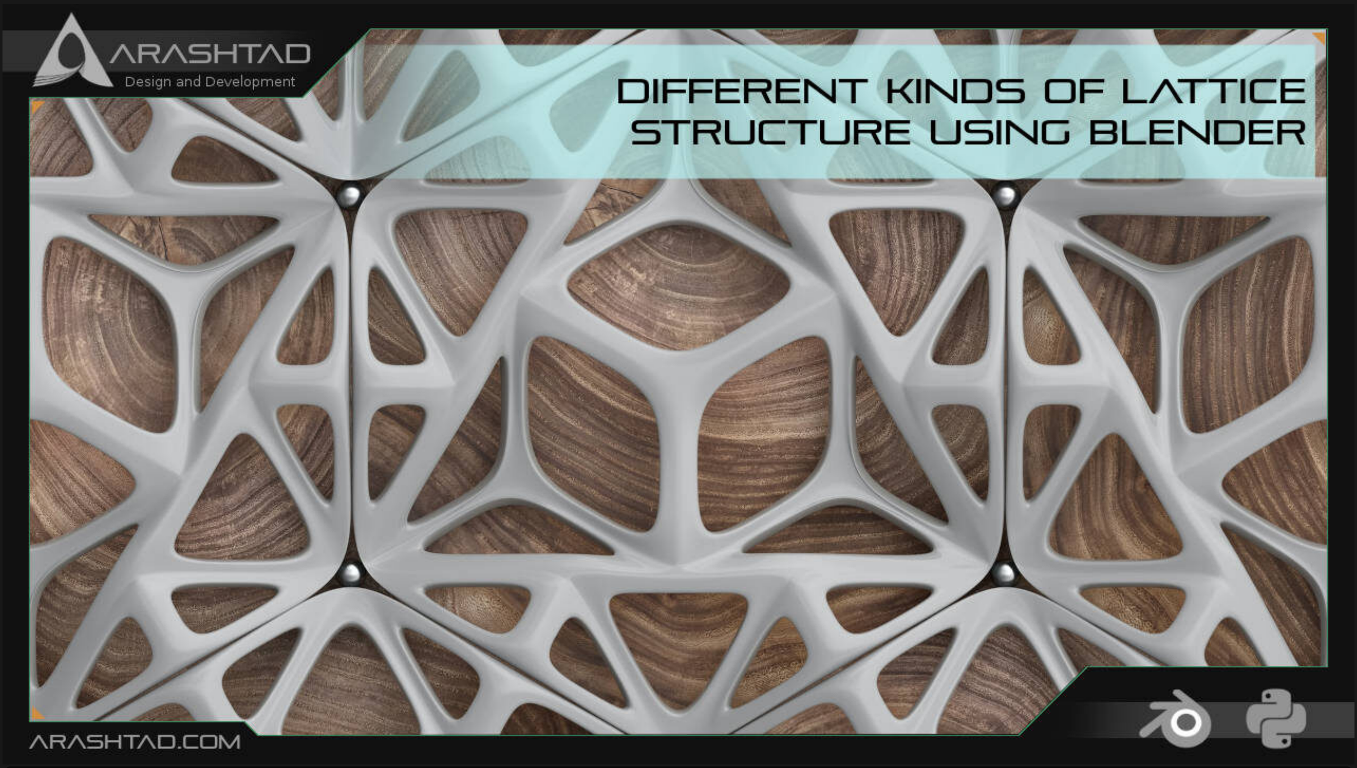
Different Kinds of Lattice Structure Using Blender

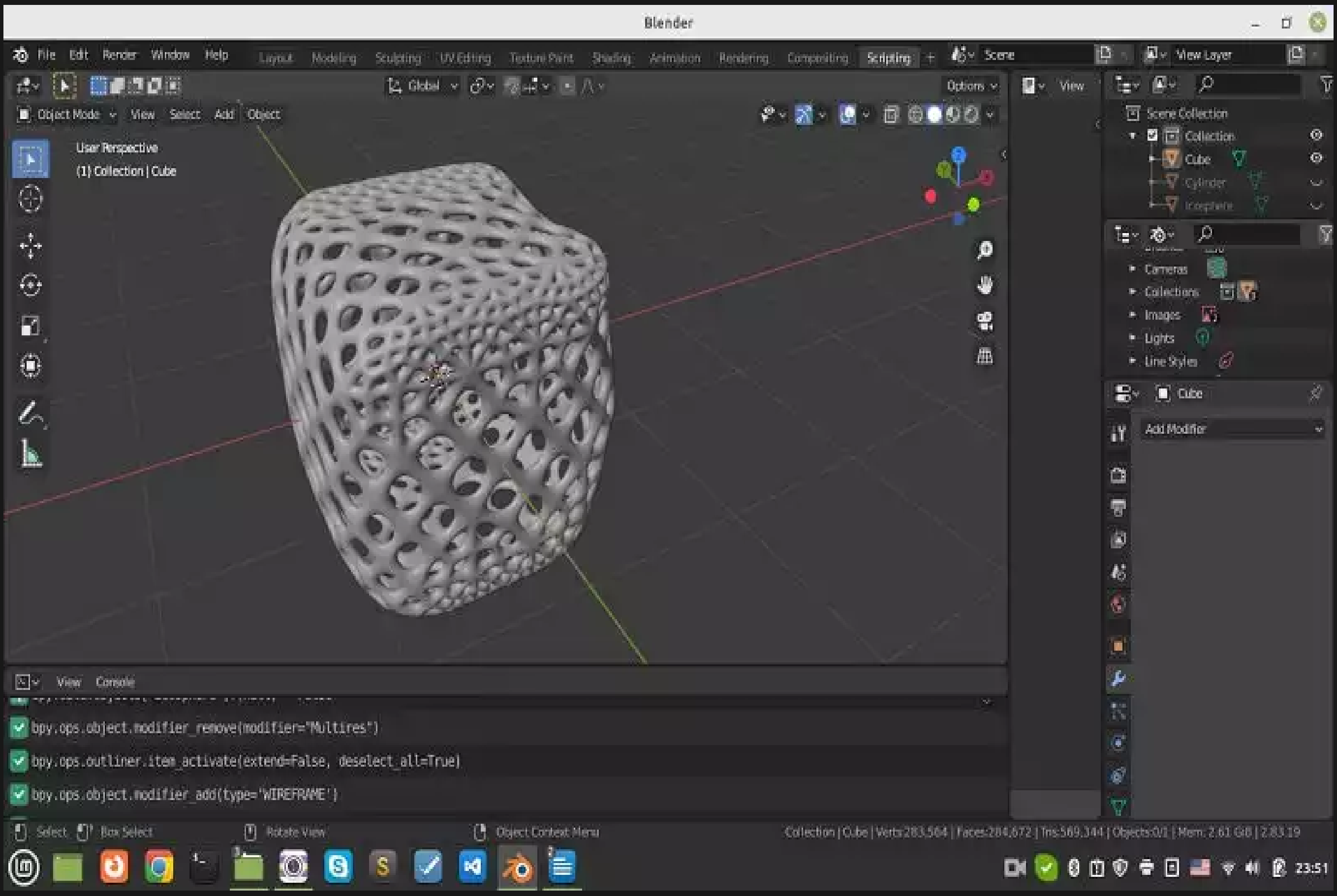
No comments



In this article, we are going to directly use the Blender modifiers to create an aesthetic costume lattice structure in Blender. But, as we have said before, the purpose of creating these kinds of lattice structures is art and aesthetics, because the end result is more stochastic rather than deterministic. We start with adding a simple shape like a cube, cylinder, or cone and use modifiers to change the formation of the mesh, then use the Wireframe modifier to make a lattice form out of our mesh, after that, by using other modifiers like Multiresolution or smooth, we post-process our design.

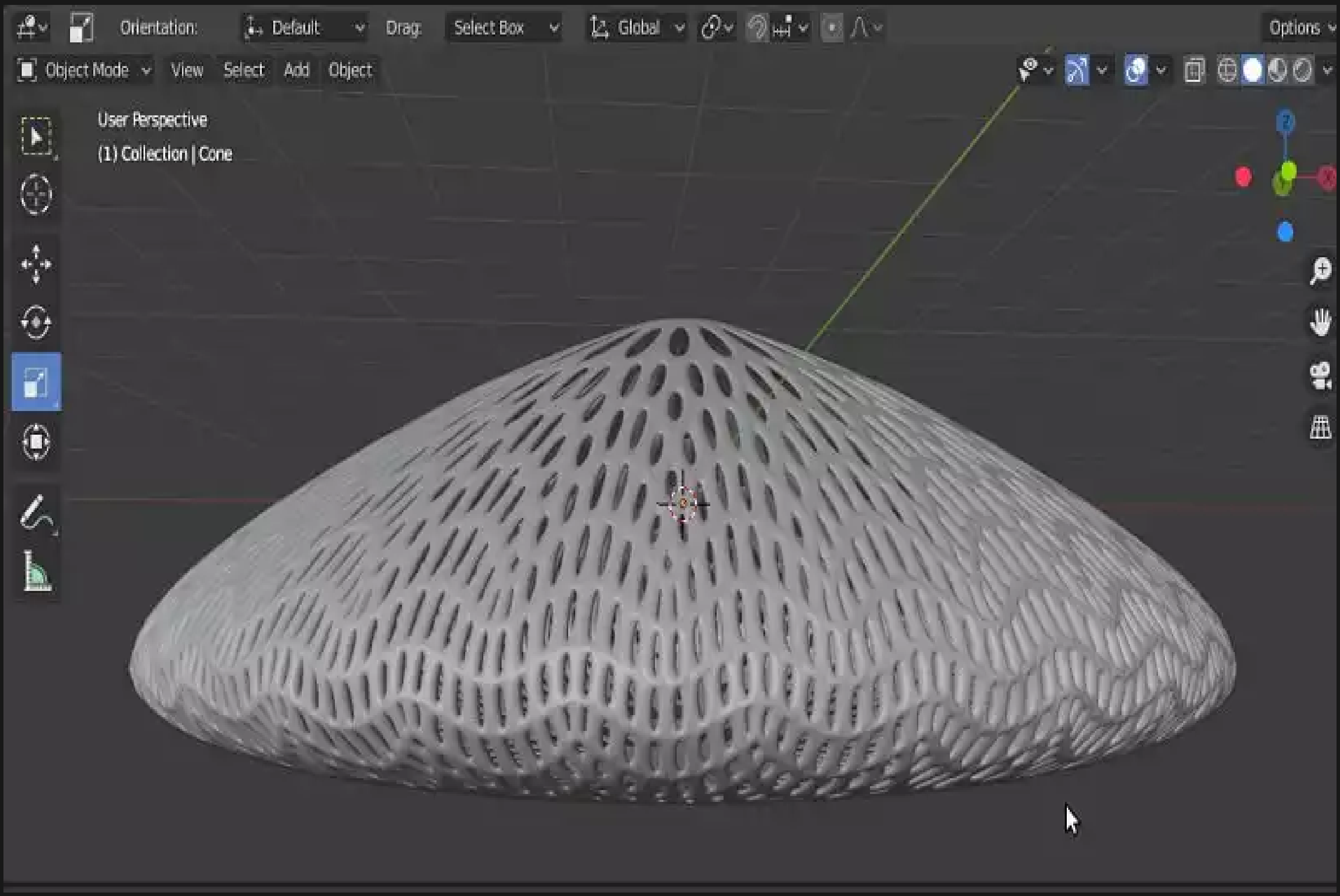
Using Blender Modifiers to Create A Lattice Structure

In this 3rd part of our tutorial, we are going to directly use the Blender modifiers to create an aesthetic costume lattice structure but as we have said before the purpose of creating this kind of lattice structure is art and aesthetics, because the end result is more stochastic rather than deterministic.


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | Cube
Scene Collection
Collection
Cube
Cylinder
Icosphere
Cameras
Collections
Images
Lights
Line Styles
Cube
Add Modifier
View Console
Select Box Select Rotate View Object Context Menu

```
bpy.ops.object.modifier_remove(modifier="Multires")
bpy.ops.outliner.item_activate(extend=False, deselect_all=True)
bpy.ops.object.modifier_add(type='WIREFRAME')
```

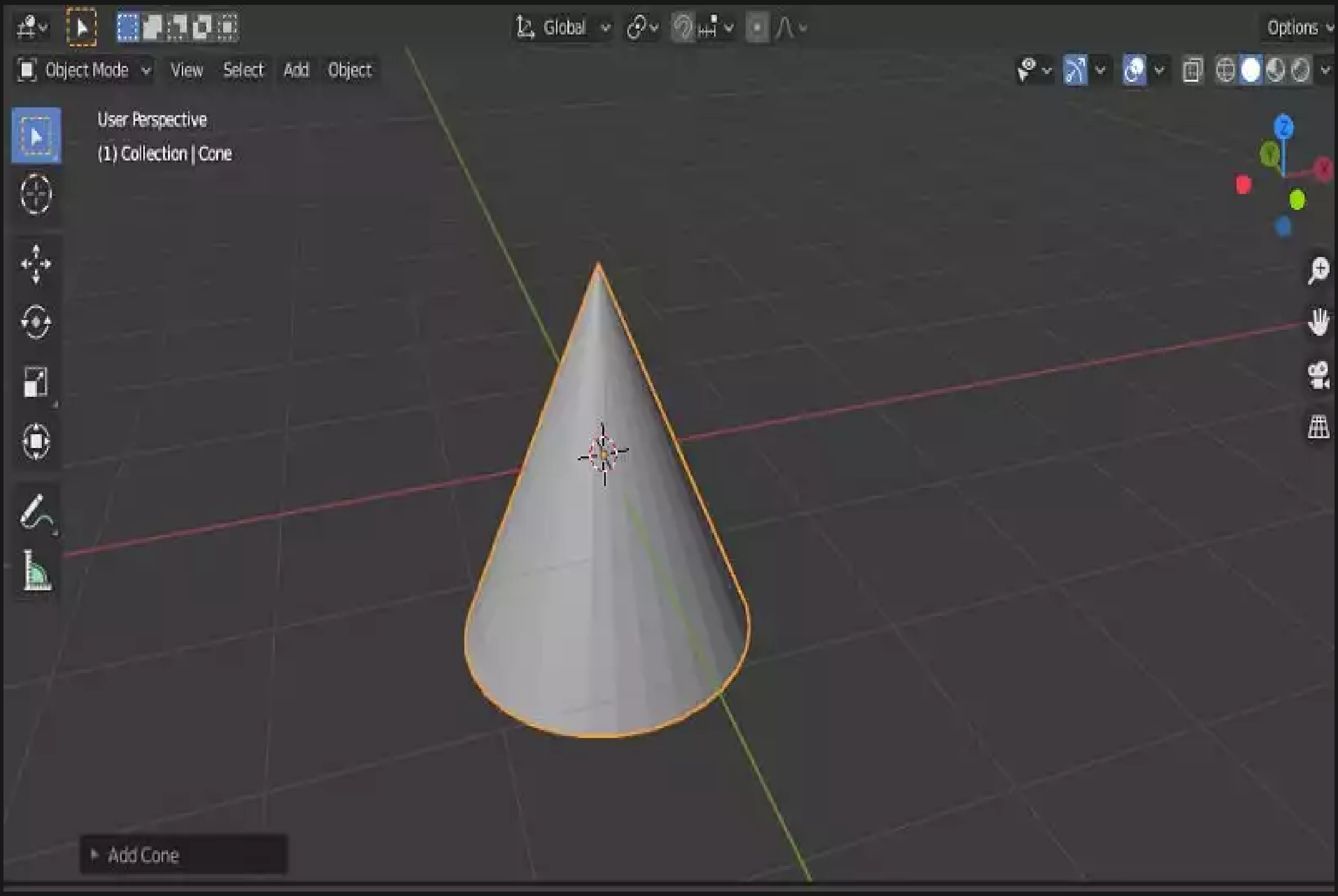
```
Collection | Cube | Verts:283,564 | Faces:284,672 | Tris:569,344 | Objects:0/1 | Mem: 2.61 GiB | 2.83.19
```



We start with adding a simple shape like a cube, cylinder, or cone and use modifiers to change the formation of the mesh, then use the Wireframe modifier to make a lattice form out of our mesh, after that, by using other modifiers like Multiresolution or smooth, we post-process our design. With that said, let's get started:

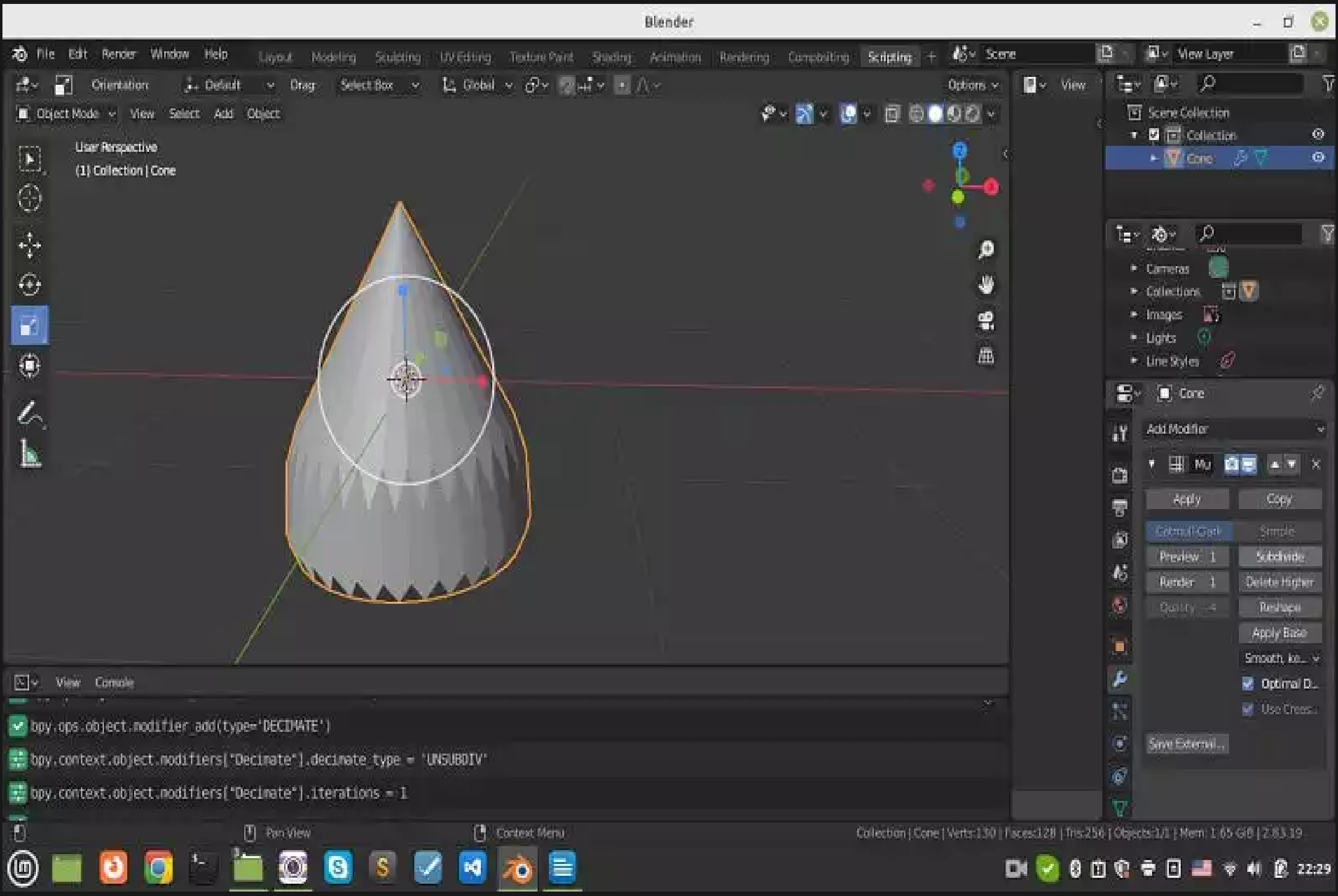
1. creating a cone:

Click on add button, on top of the page and go to `mesh >> cone`. And you can see that a cone like below is created:



2. Modifying the object:

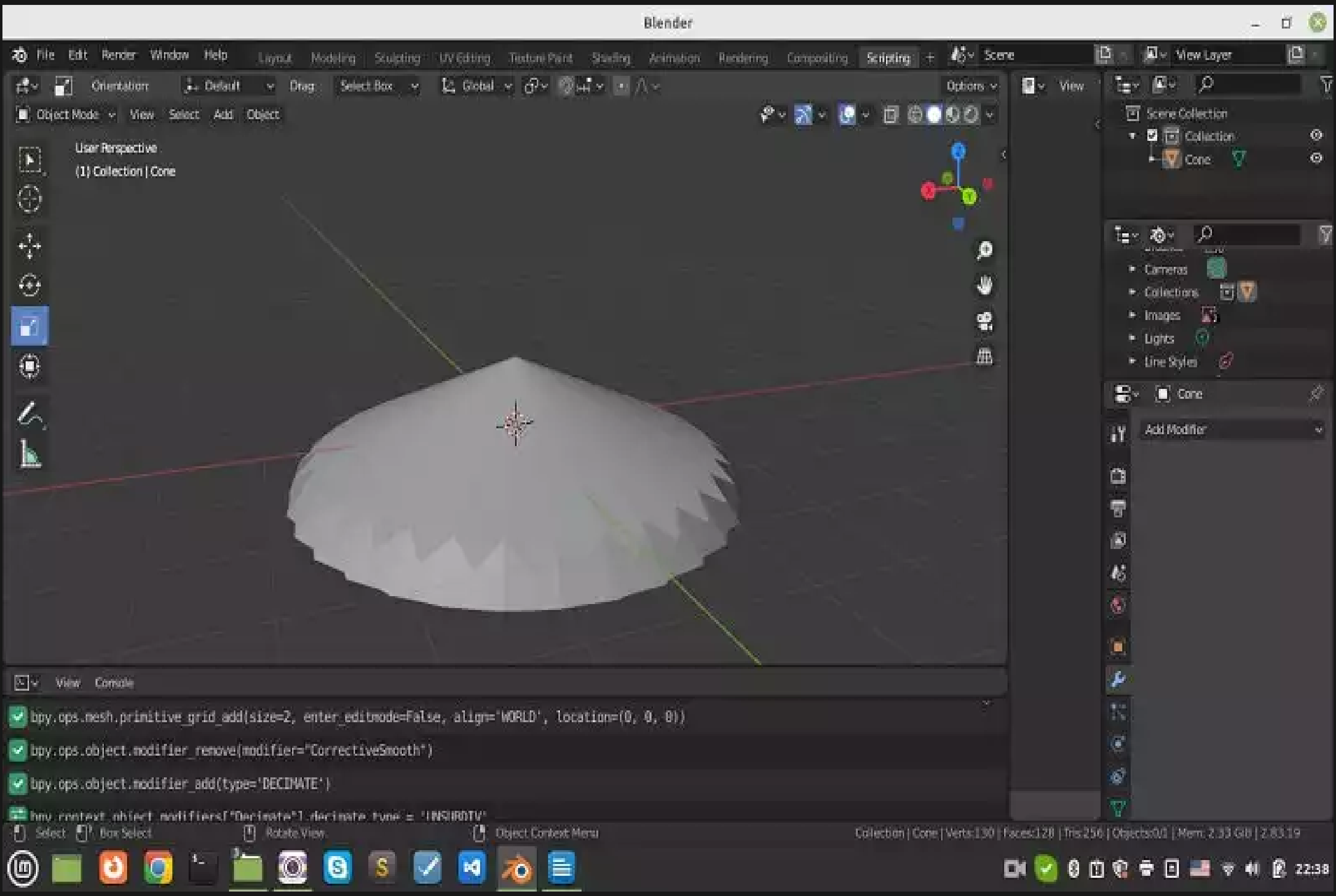
In the modifiers section, choose the multiresolution modifier and by using `Multiresulotion >> Subdivide`, you can subdivide (increase) the number of meshes on the object like the below photo:


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cone
Scene Collection
Collection
Cone
Add Modifier
Apply Copy
Subdivide
Delete Higher
Reshape
Apply Base
View Console
Collection | Cone | Verts:130 | Faces:128 | Tris:256 | Objects:1/1 | Mem: 1.65 GiB | 2.83.19

```
bpy.ops.object.modifier_add(type='DECIMATE')
bpy.context.object.modifiers["Decimate"].decimate_type = 'UNSUBDIV'
bpy.context.object.modifiers["Decimate"].iterations = 1
```

3. Scaling:

Now, scale the the modified cone in x-y direction To get a shape like below:

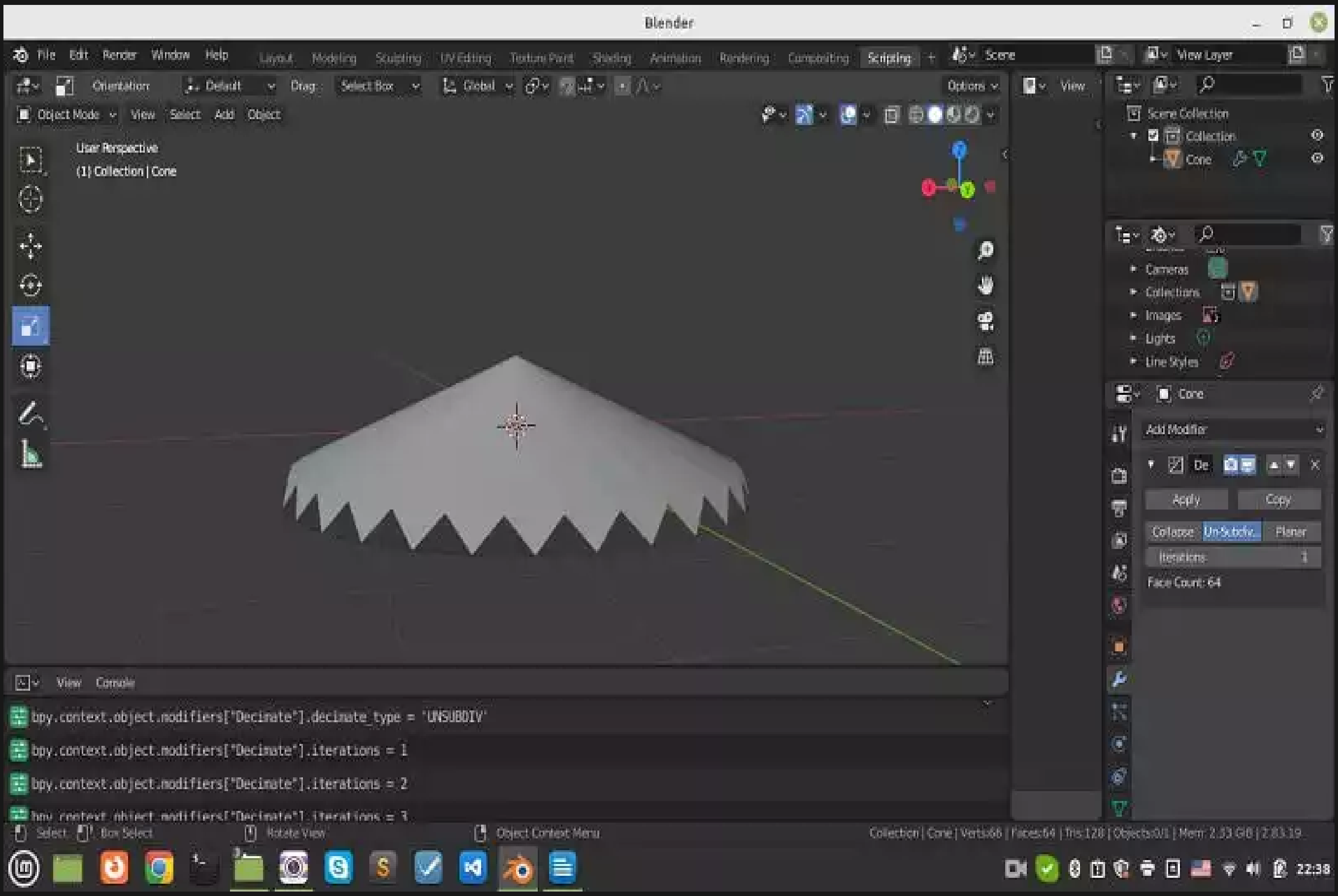

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cone
Scene Collection
Collection
Cone
Add Modifier
View Console

```
bpy.ops.mesh.primitive_grid_add(size=2, enter_editmode=False, align='WORLD', location=(0, 0, 0))
bpy.ops.object.modifier_remove(modifier="CorrectiveSmooth")
bpy.ops.object.modifier_add(type='DECIMATE')
bpy.context.object.modifiers["Decimate"].decimate_type = 'UNSUBDIV'
```

```
Collection | Cone | Verts:130 | Faces:128 | Tris:256 | Objects:0/1 | Mem: 2.33 GiB | 2.83.19
```

4. Decreasing the number of meshes:

In the modifiers section, select the Decimate modifier and by using the `Decimate >> Unsubdivide`, decrease the number of meshes:


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cone
Scene Collection
Collection
Cone
Add Modifier
Apply
Copy
View Console

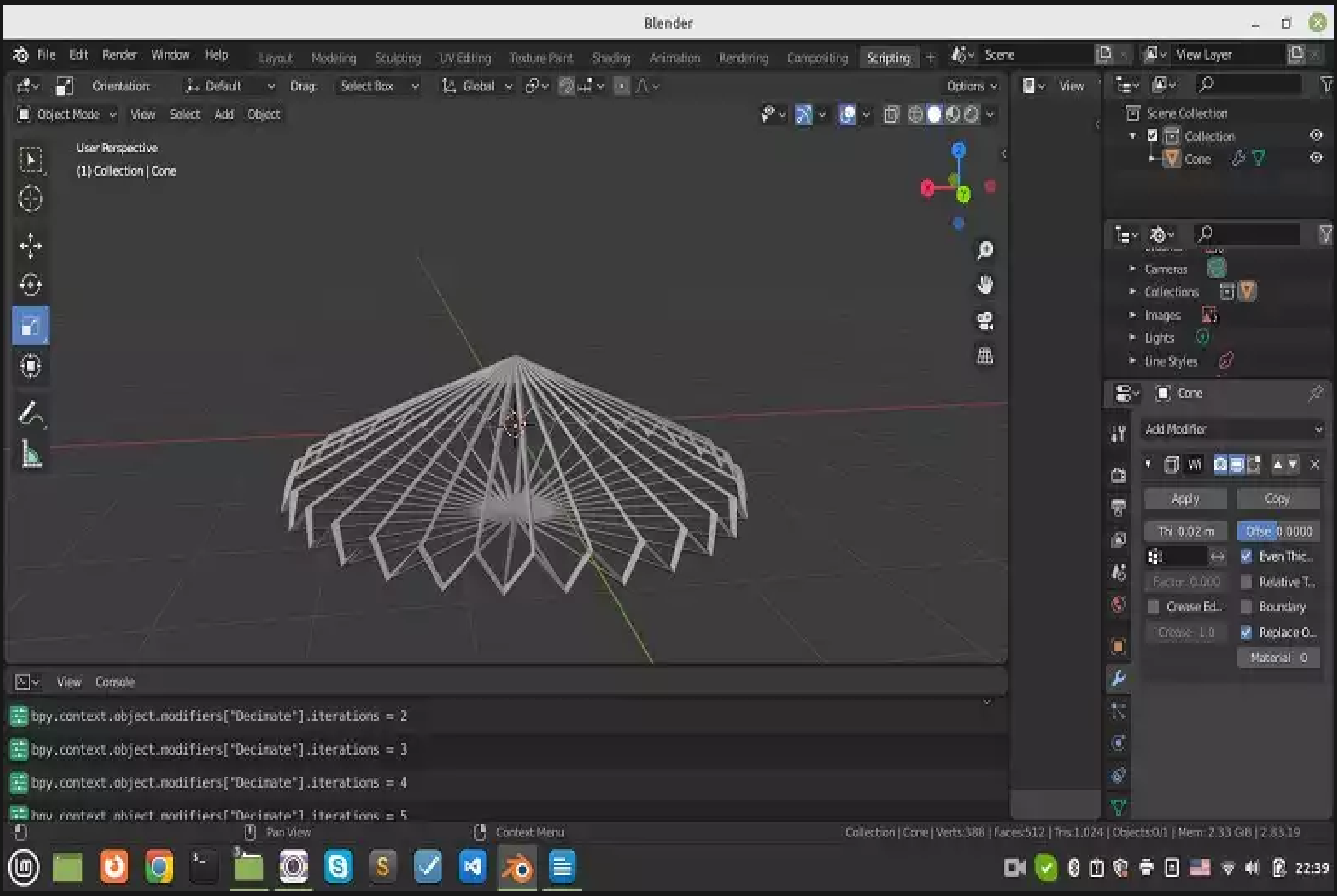
```
bpy.context.object.modifiers["Decimate"].decimate_type = 'UNSUBDIV'
bpy.context.object.modifiers["Decimate"].iterations = 1
bpy.context.object.modifiers["Decimate"].iterations = 2
bpy.context.object.modifiers["Decimate"].iterations = 3
```

Collapse | Un-Subdiv | Planar
Iterations 1
Face Count: 64

Collection | Cone | Verts:66 | Faces:64 | Tris:128 | Objects:0/1 | Mem: 2.33 GiB | 2.83.19

5. Creating a lattice form:

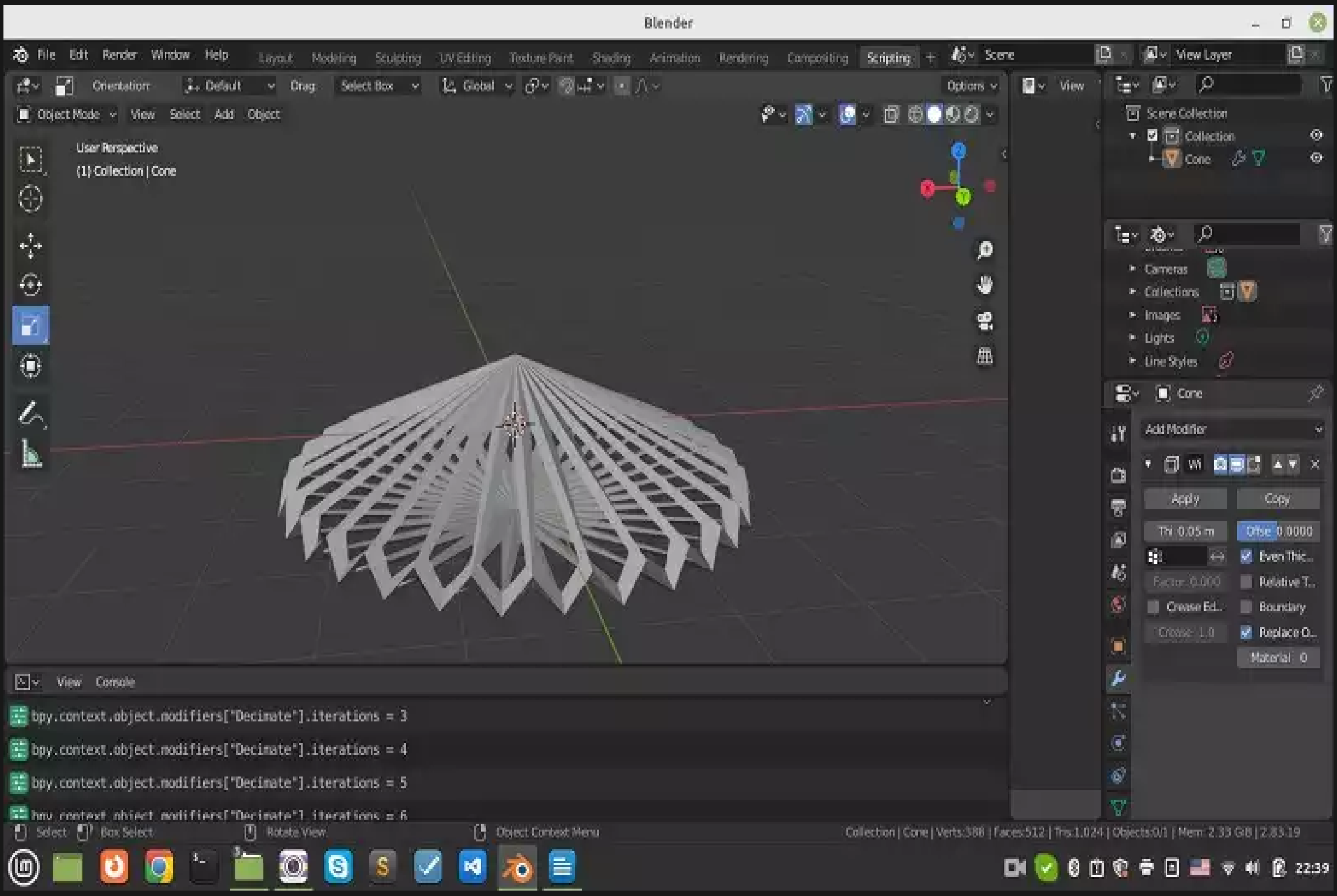
In the modifiers section, find the wireframe and use it to create lattice form of the object like below:


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Scene
View Layer
Object Mode View Select Add Object
User Perspective
(1) Collection | Cone
Scene Collection
Collection
Cone
Add Modifier
Apply
Copy
Thi 0.02 m
Offse 0.0000
Even Thic...
Relative T...
Boundary
Crease Ed...
Replace O...
Material 0
View Console
Collection | Cone | Verts:388 | Faces:512 | Tris:1,024 | Objects:0/1 | Mem: 2.33 GiB | 2.83.19

```
bpy.context.object.modifiers["Decimate"].iterations = 2
bpy.context.object.modifiers["Decimate"].iterations = 3
bpy.context.object.modifiers["Decimate"].iterations = 4
bpy.context.object.modifiers["Decimate"].iterations = 5
```

6. Finding the right thickness:

Change the thickness of the wireframe according to the size of the object. This part is a little bit tricky because if they are too thin, the wireframe will shrink at the time of smoothing and if they are too thick, they may not create a nice structure or holes may shrink instead of the wireframes.

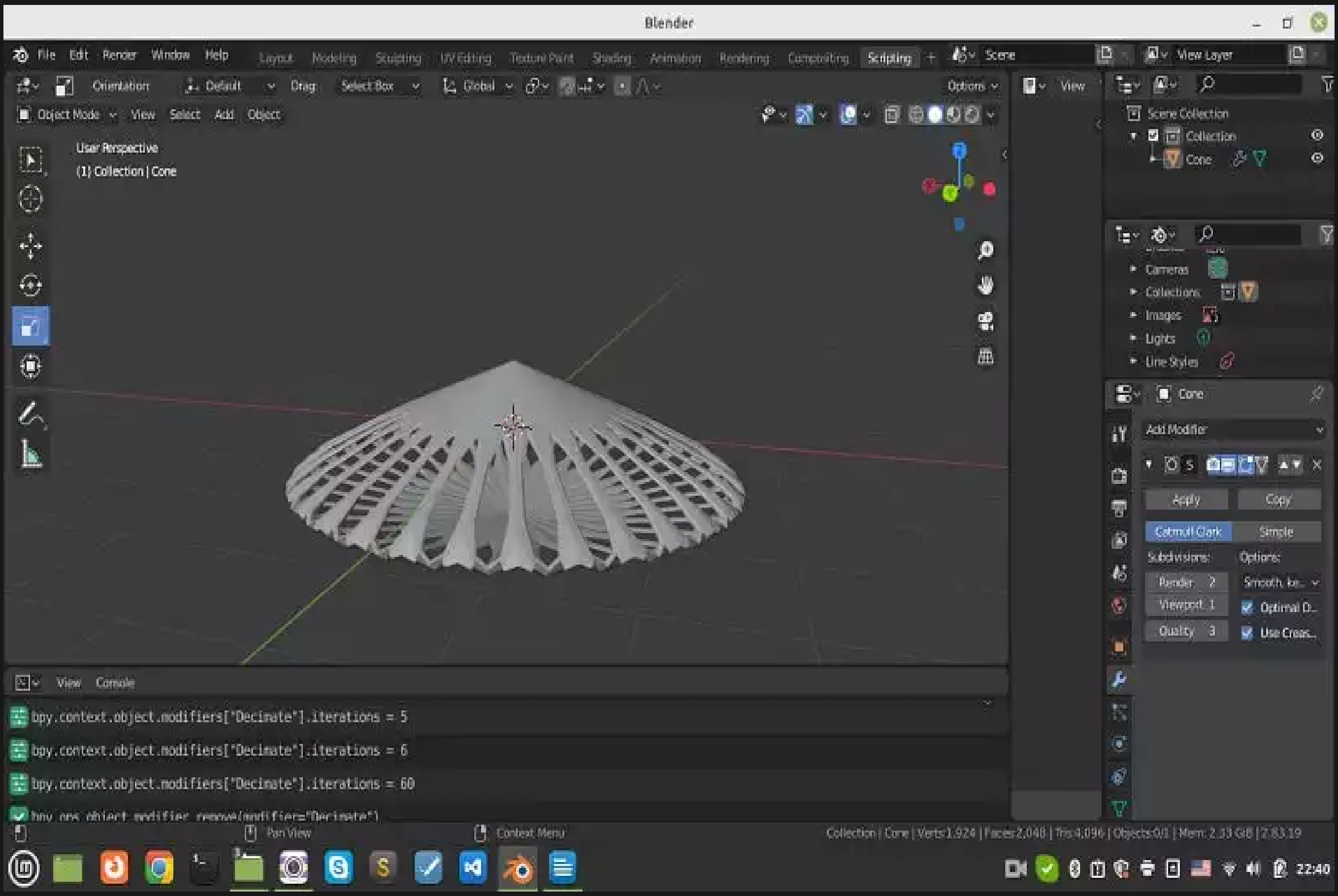


```
bpy.context.object.modifiers["Decimate"].iterations = 3
bpy.context.object.modifiers["Decimate"].iterations = 4
bpy.context.object.modifiers["Decimate"].iterations = 5
bpy.context.object.modifiers["Decimate"].iterations = 6
```

Collection | Cone | Verts:388 | Faces:512 | Tris:1,024 | Objects:0/1 | Mem: 2.33 GiB | 2.83.19

7. Smoothing the object:

Use the Multiresolution modifier once again to make the object smoother:



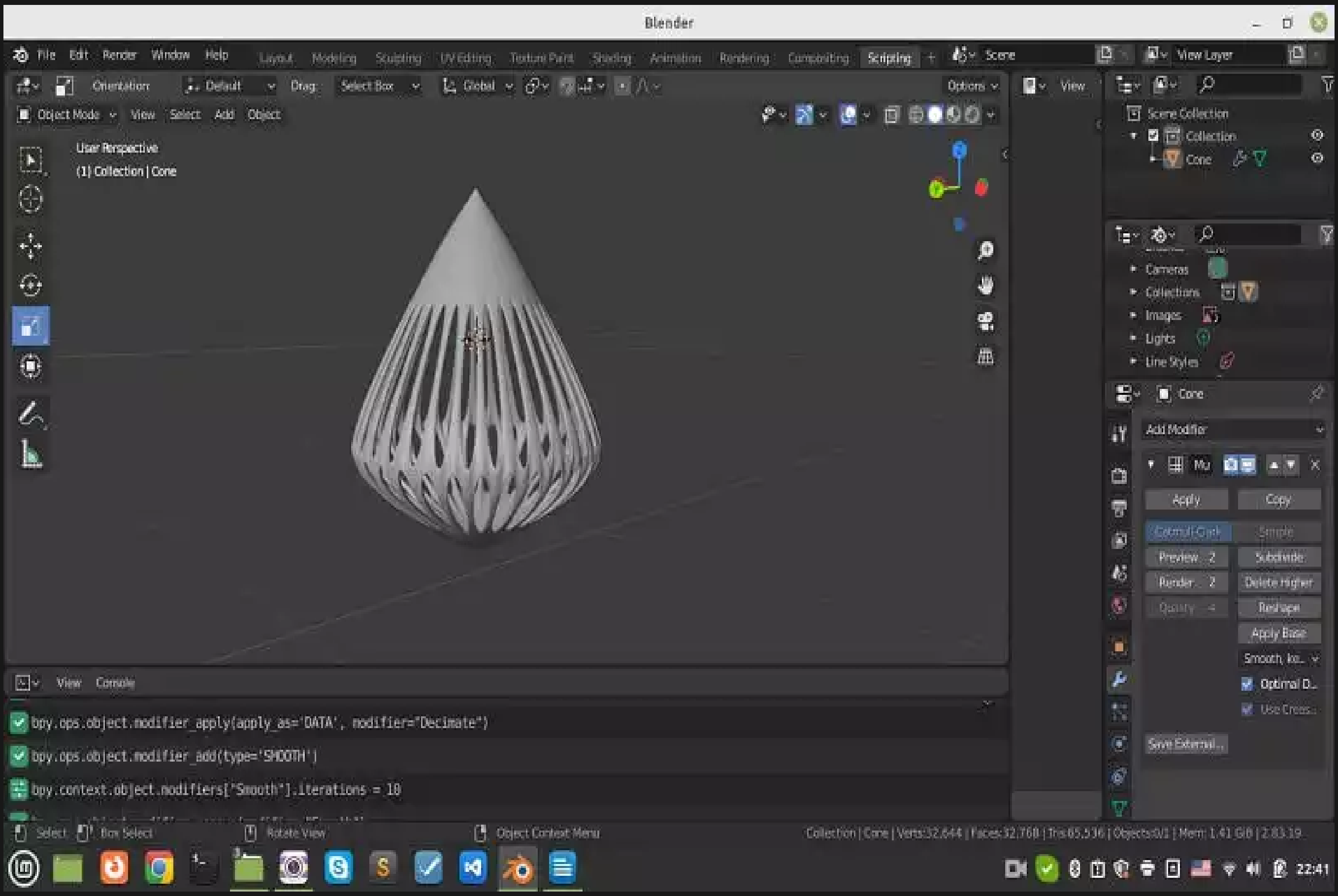
Catmull-Clark | Simple
Subdivisions: Render 2, Viewport 1, Quality 3
Options: Smooth, ke..., ✓ Optimal D..., ✓ Use Creas...

```
bpy.context.object.modifiers["Decimate"].iterations = 5
bpy.context.object.modifiers["Decimate"].iterations = 6
bpy.context.object.modifiers["Decimate"].iterations = 60
bpy.ops.object.modifier_remove(modifier="Decimate")
```

Collection | Cone | Verts:1,924 | Faces:2,048 | Tris:4,096 | Objects:0/1 | Mem: 2.33 GiB | 2.83.19

8. Final scaling:

Scale the object according to your design preferences.



```
bpy.ops.object.modifier_apply(apply_as='DATA', modifier="Decimate")
bpy.ops.object.modifier_add(type='SMOOTH')
bpy.context.object.modifiers["Smooth"].iterations = 10
```

Collection | Cone | Verts:32,644 | Faces:32,768 | Tris:65,536 | Objects:0/1 | Mem: 1.41 GiB | 2.83.19

Notice that the procedure of designing the lattice structure is not the same for all the objects. It is very optional and based on your taste which method to choose.

In the next part, we are going to create a lattice structure using Blender for a different object with a few differences in the procedure.

Tricky Parts of Designing Lattice Structures in Blender

In this section, we are going to directly use the Blender modifiers to create aesthetic costume lattice structures but as we have said before the purpose of creating this kind of lattice structure is art and aesthetics, because the end result is more stochastic rather than deterministic. The main focus of this section is to teach you the tricky parts of designing these structures.

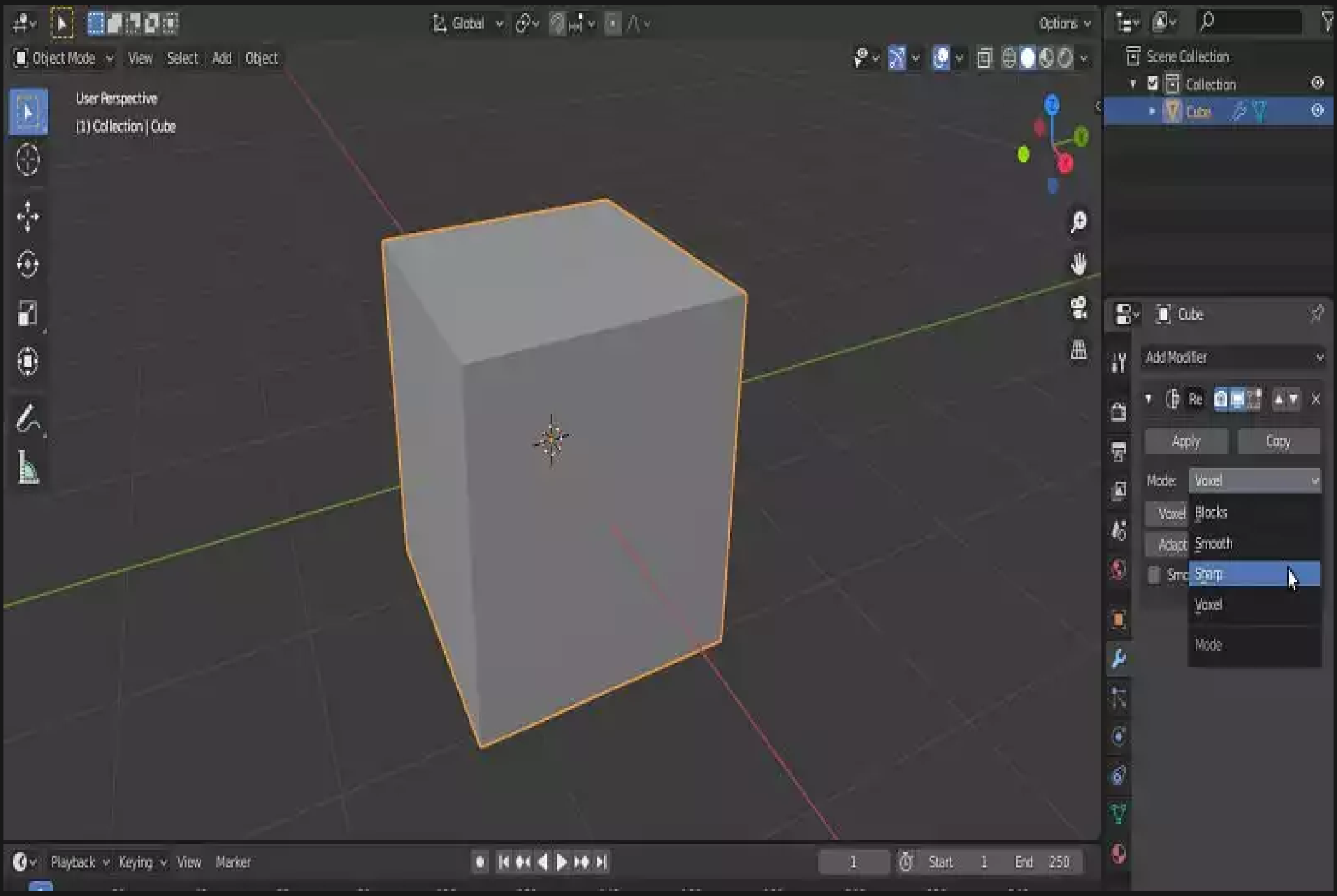
Mesh structures:

The most important part of designing every lattice structure is the mesh formation and there are many ways to change the way the meshes are formed. Different mesh formations are as follows:

1. Triangular mesh
2. Voxel mesh
3. Quadrimesh
4. Blocks
5. Smooth

And so on.

The trickiest part about designing lattice structures is finding the best option for forming the meshes.

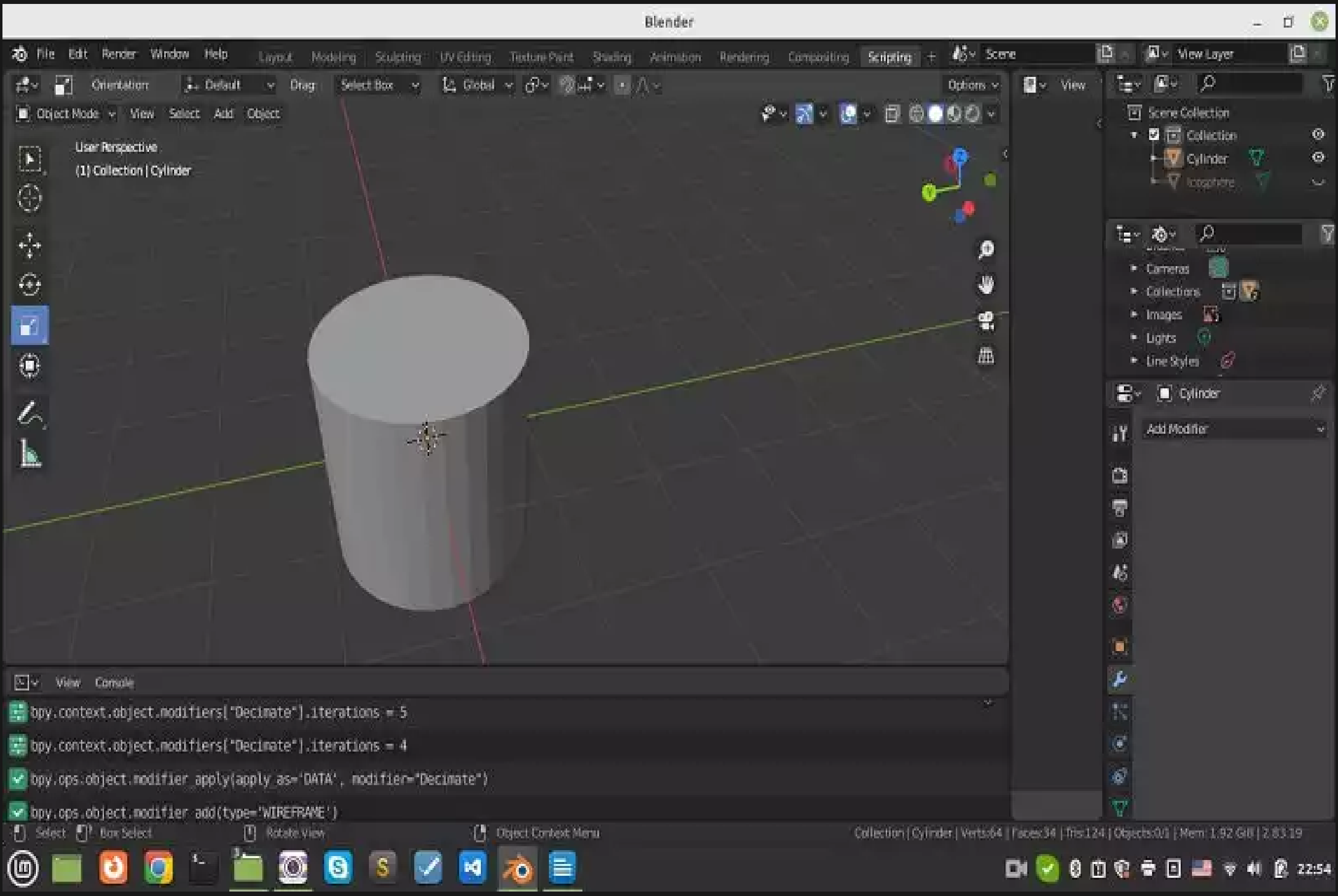


Lattice Structure for A Clyinder

The next design we are going to work on is the lattice structure for a cylinder. The recommended steps are as follows:

1. Creating a cylinder:

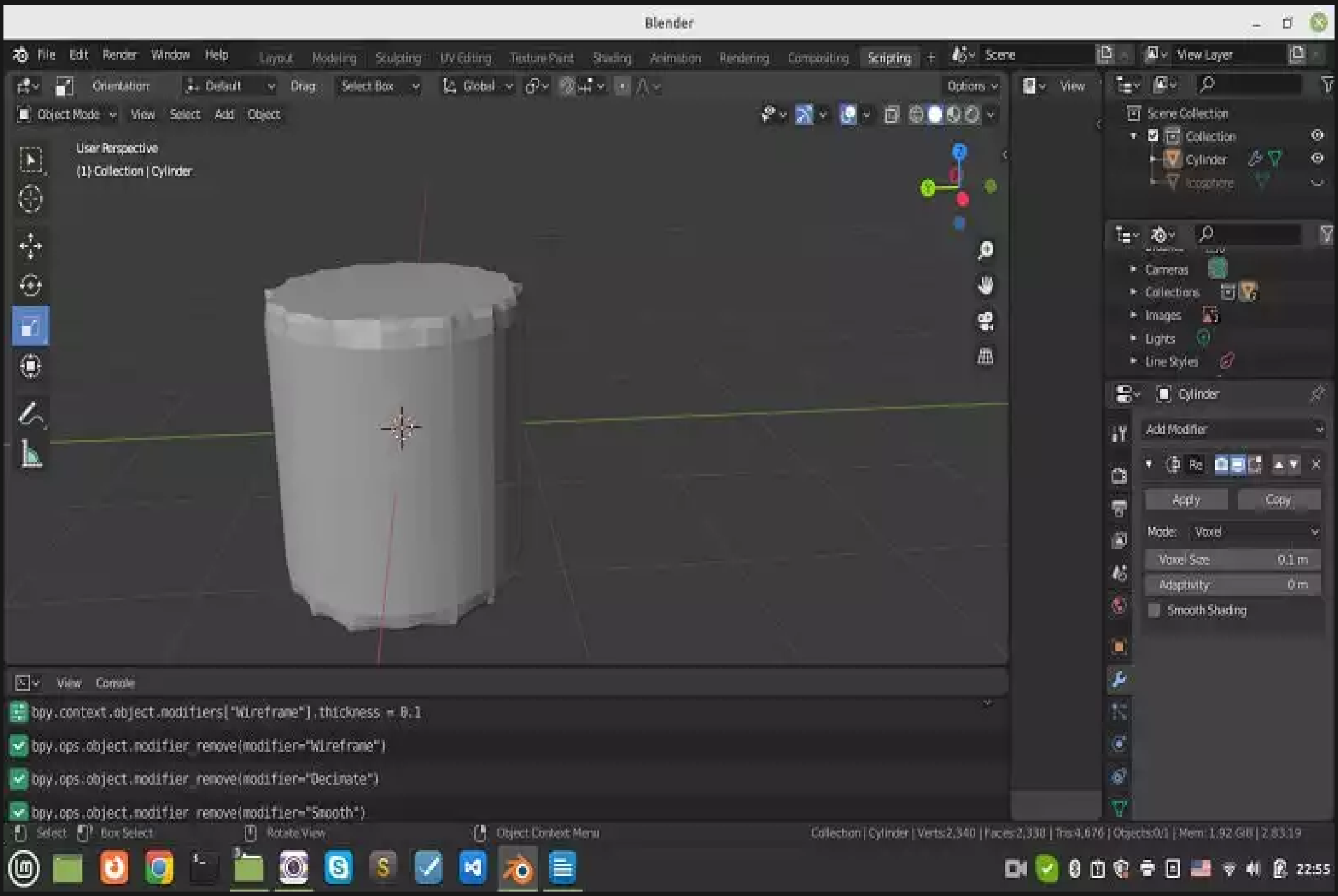
From the top of the page, click on `Add >> Mesh >> Cylinder` and you will be able to see that we have a cylinder created in the origin.



```
bpy.context.object.modifiers["Decimate"].iterations = 5
bpy.context.object.modifiers["Decimate"].iterations = 4
bpy.ops.object.modifier_apply(apply_as='DATA', modifier="Decimate")
bpy.ops.object.modifier_add(type='WIREFRAME')
```

2. Remesh the object in voxel form:

By using the Remesh modifier from the modifiers section, you can change the formation of the meshes from triangular to voxel-based.


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cylinder
Scene Collection
Collection
Cylinder
Icosphere
Cameras
Collections
Images
Lights
Line Styles
Add Modifier
Apply
Copy
View Console

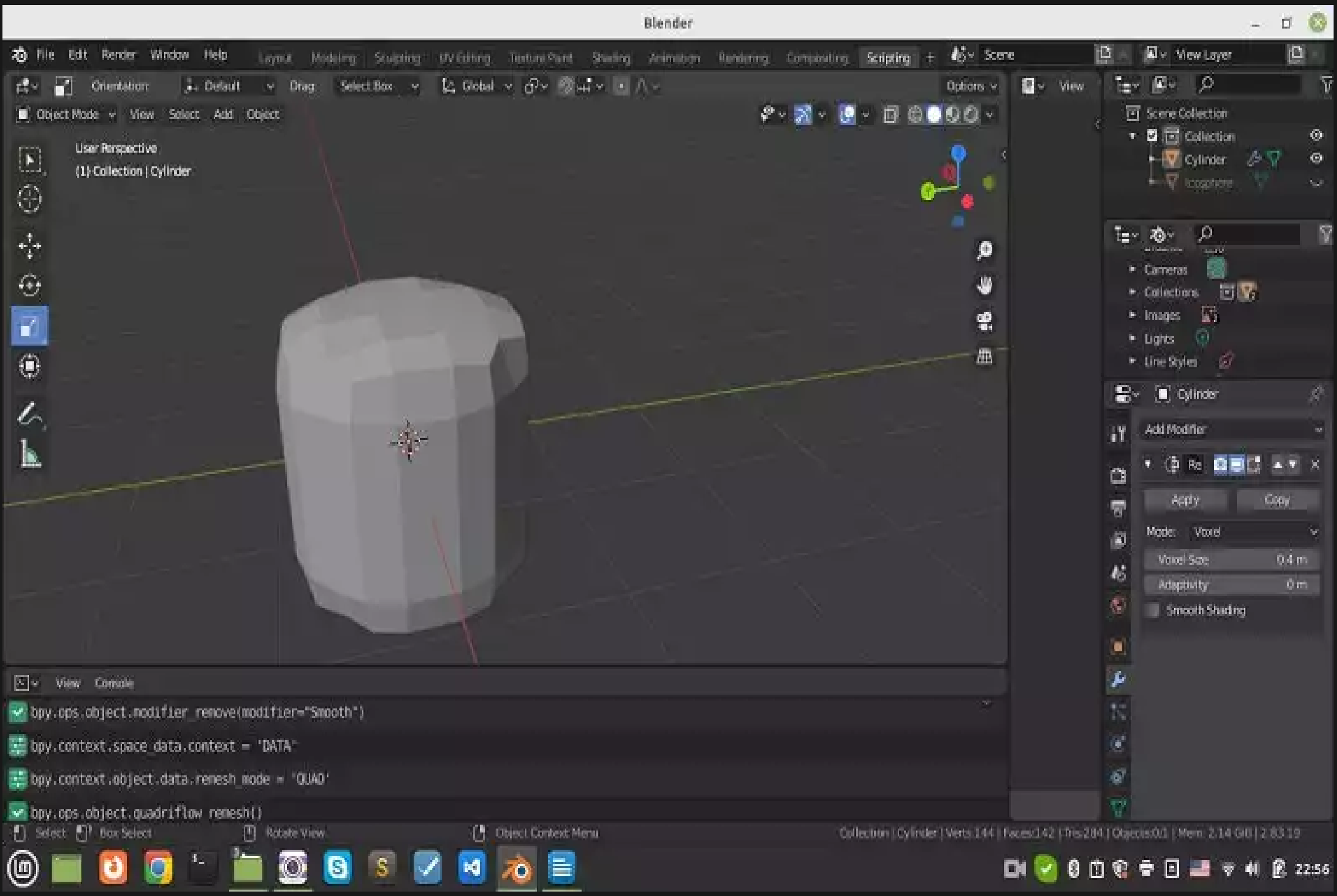
Mode: Voxel
Voxel Size 0.1 m
Adaptivity 0 m
Smooth Shading

```
bpy.context.object.modifiers["Wireframe"].thickness = 0.1
bpy.ops.object.modifier_remove(modifier="Wireframe")
bpy.ops.object.modifier_remove(modifier="Decimate")
bpy.ops.object.modifier_remove(modifier="Smooth")
```

Collection | Cylinder | Verts:2,340 | Faces:2,338 | Tris:4,676 | Objects:0/1 | Mem: 1.92 GiB | 2.83.19

3. Change the size of the voxels:

By selecting the remesh modifier from the modifiers section, you can remesh the object from triangular to voxel-based and also decrease the number of meshes leading to making it larger.



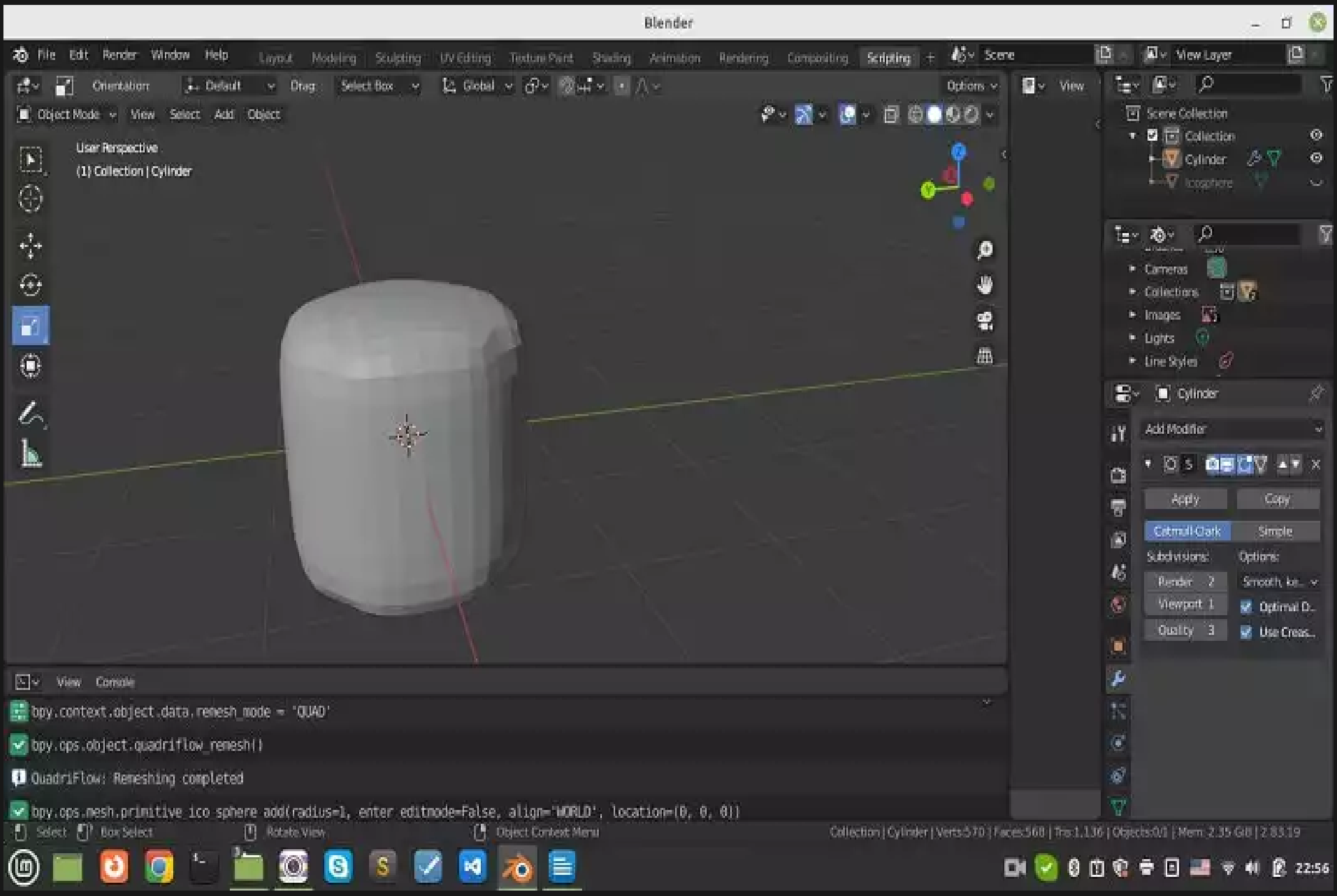
Mode: Voxel
Voxel Size 0.4 m
Adaptivity 0 m
Smooth Shading

```
bpy.ops.object.modifier_remove(modifier="Smooth")
bpy.context.space_data.context = 'DATA'
bpy.context.object.data.remesh_mode = 'QUAD'
bpy.ops.object.quadriflow_remesh()
```

Collection | Cylinder | Verts:144 | Faces:142 | Tris:284 | Objects:0/1 | Mem: 2.14 GiB | 2.83.19

4. Subdivide the mesh

Using the Multi-resolution modifier, make the object smoother.

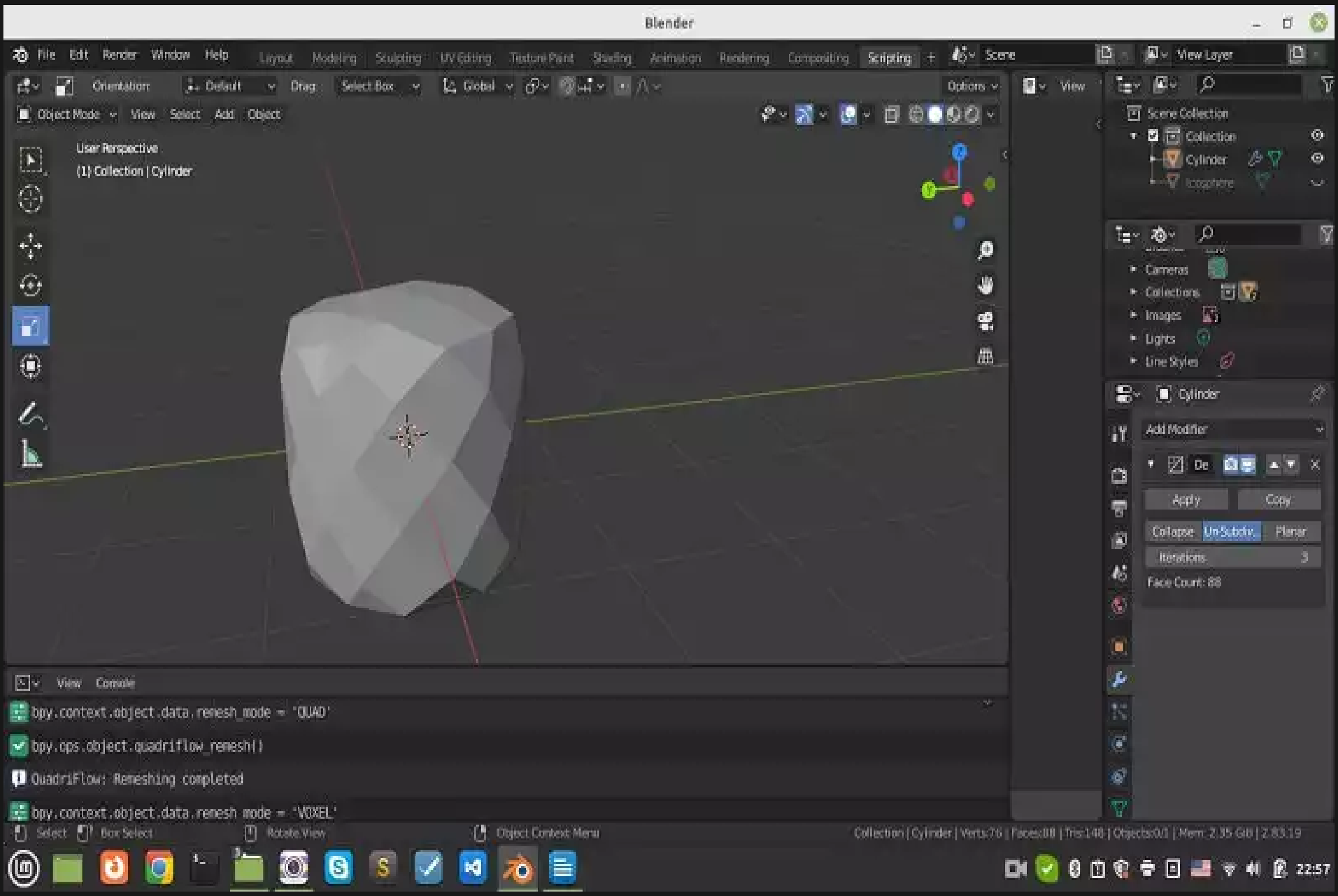

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cylinder
Scene Collection
Collection
Cylinder
Icosphere
Add Modifier
Apply Copy
Catmull-Clark Simple
Subdivisions: Options:
Render 2
Viewport 1
Quality 3

```
bpy.context.object.data.remesh_mode = 'QUAD'
bpy.ops.object.quadriflow_remesh()
QuadriFlow: Remeshing completed
bpy.ops.mesh.primitive_ico_sphere_add(radius=1, enter_editmode=False, align='WORLD', location=(0, 0, 0))
```

Collection | Cylinder | Verts:570 | Faces:568 | Tris:1,136 | Objects:0/1 | Mem: 2.35 GiB | 2.83.19

5. Unsubdivide the mesh:

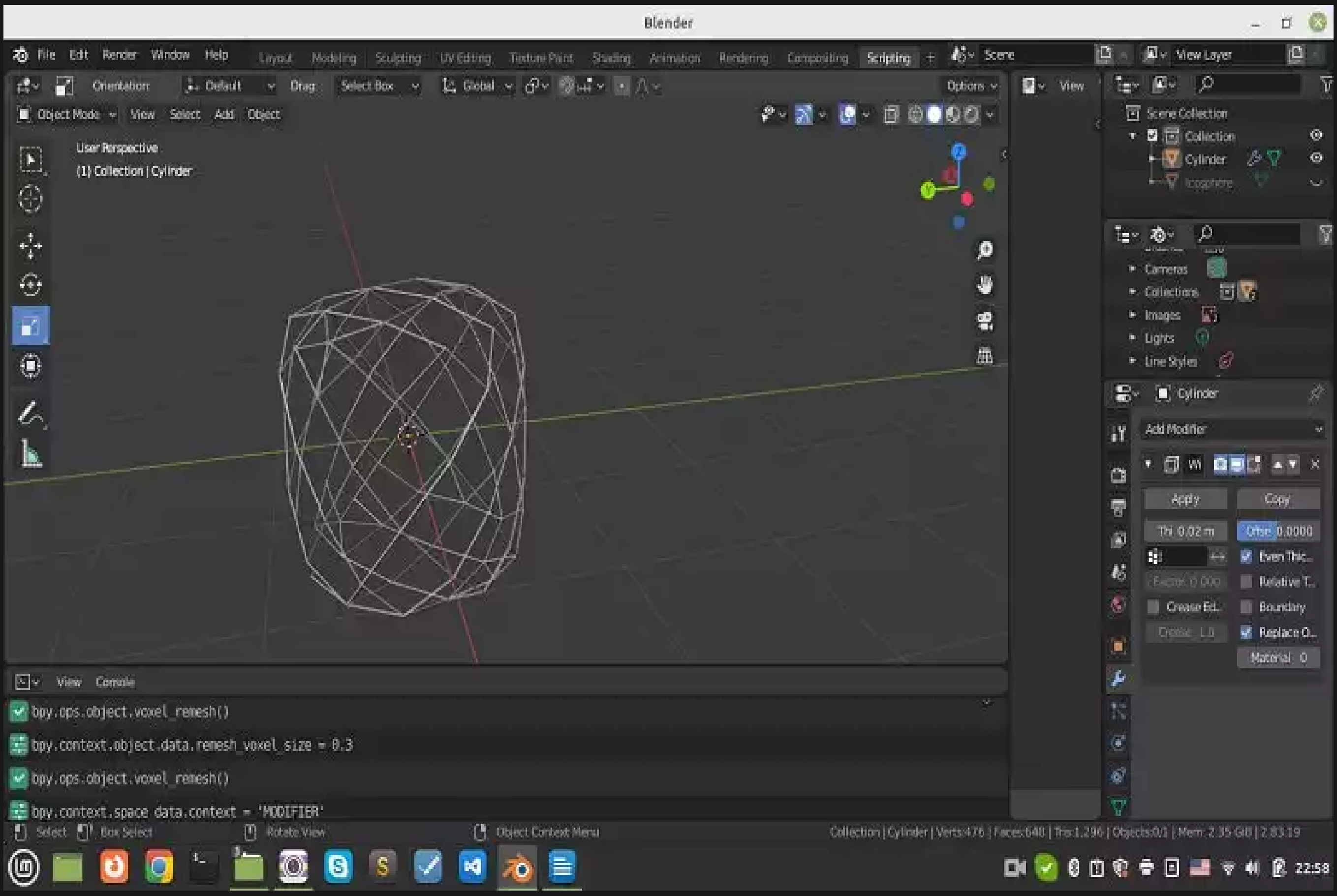
Using the decimate modifier, unsubdivided the mesh so that we will have larger holes in our lattice structure. The reason why we subdivide and then unsubdivided is that the shape of the meshes changes and becomes more beautiful.


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cylinder
Scene Collection
Collection
Cylinder
Icosphere
Cameras
Collections
Images
Lights
Line Styles
Add Modifier
Apply Copy
Collapse Un-Subdiv. Planar
Iterations 3
Face Count: 88
View Console

```
bpy.context.object.data.remesh_mode = 'QUAD'
bpy.ops.object.quadriflow_remesh()
QuadriFlow: Remeshing completed
bpy.context.object.data.remesh_mode = 'VOXEL'
```

6. Wireframe modifier:

And then, use the Wireframe modifier to form the lattice shape.


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Object Mode View Select Add Object
User Perspective
(1) Collection | Cylinder
Scene Collection
Collection
Cylinder
Icosphere
Cameras
Collections
Images
Lights
Line Styles
Add Modifier
Apply
Copy

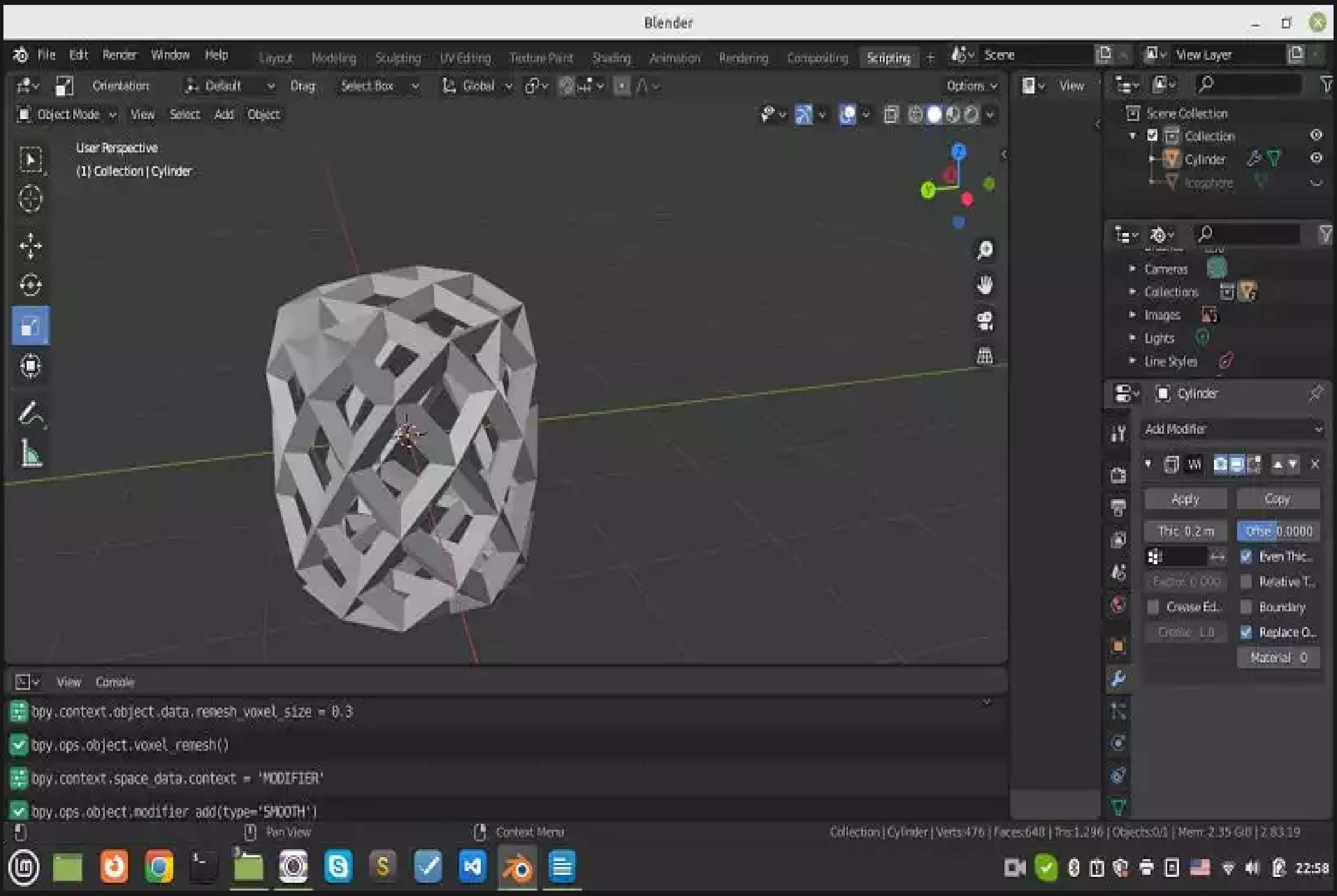
Thi 0.02 m
Offse 0.0000
Even Thic...
Factor 0.000
Relative T...
Crease Ed...
Boundary
Crease 1.0
Replace O...
Material 0

```
bpy.ops.object.voxel_remesh()
bpy.context.object.data.remesh_voxel_size = 0.3
bpy.ops.object.voxel_remesh()
bpy.context.space_data.context = 'MODIFIER'
```

Collection | Cylinder | Verts:476 | Faces:648 | Tris:1,296 | Objects:0/1 | Mem: 2.35 GiB | 2.83.19

7. thickness of the wireframe:

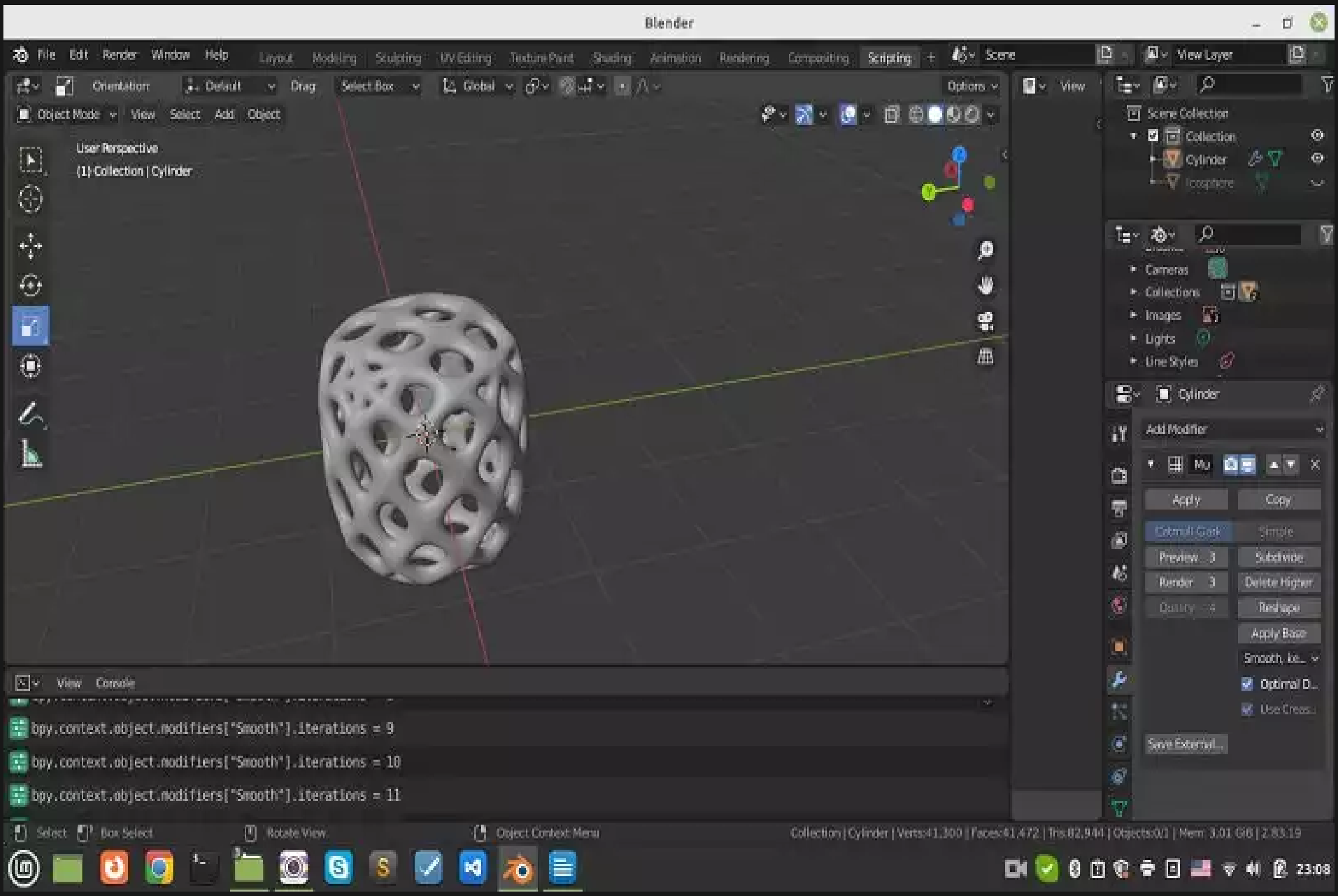
Change the thickness of the wireframe to reach the optimal shape.



```
bpy.context.object.data.remesh_voxel_size = 0.3
bpy.ops.object.voxel_remesh()
bpy.context.space_data.context = 'MODIFIER'
bpy.ops.object.modifier_add(type='SMOOTH')
```

8. Another subdivision:

Again, using the Multi-resolution modifier, subdivide the mesh to get a smooth shape.

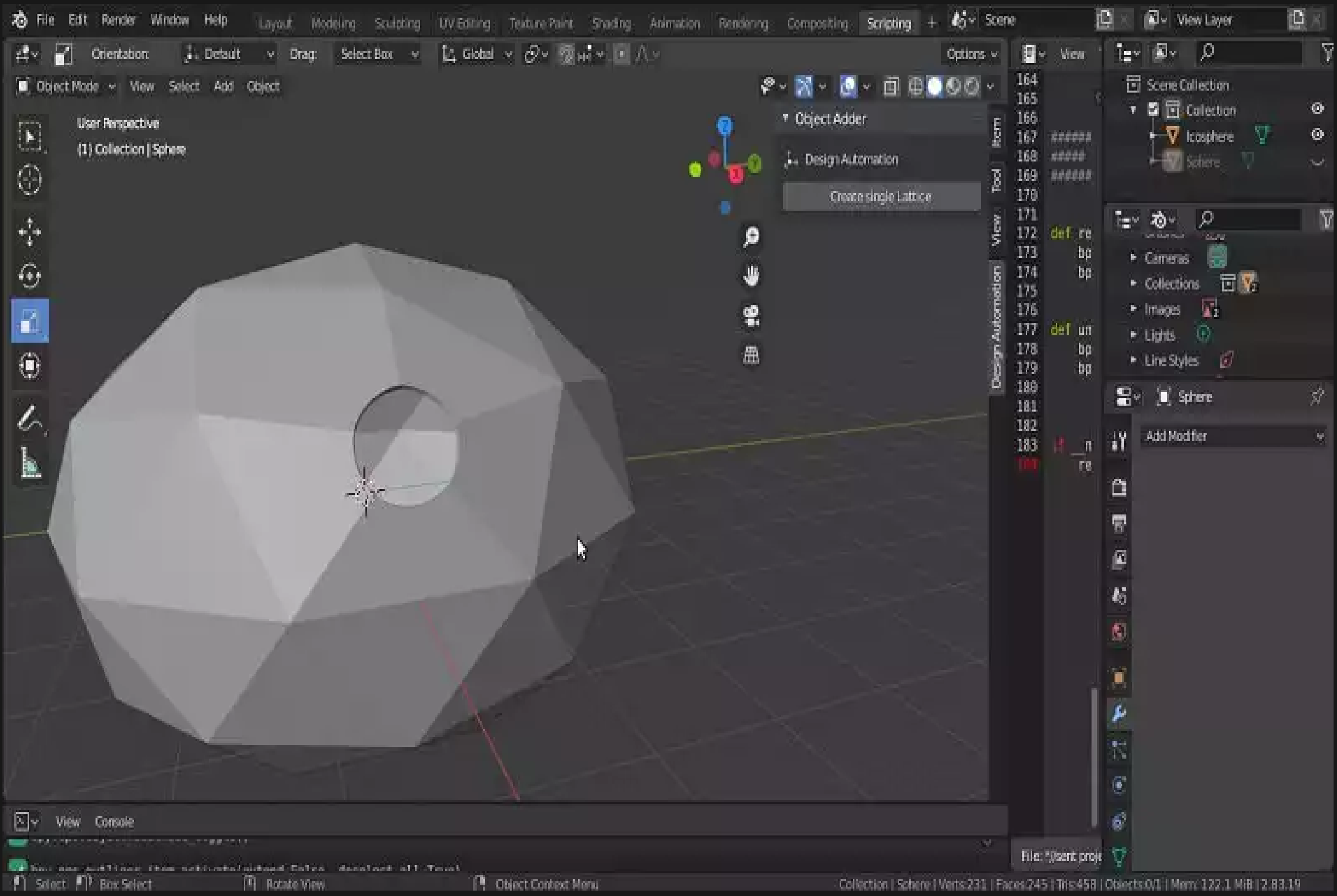


```
bpy.context.object.modifiers["Smooth"].iterations = 9
bpy.context.object.modifiers["Smooth"].iterations = 10
bpy.context.object.modifiers["Smooth"].iterations = 11
```

You might want to have your own costume lattice shape to be applied to an object. In that case, we have a more complex process of placing the lattice objects on the main object and smoothing the end result. That kind of lattice is better to be performed with the aid of python blender API. In the next chapter, we work on the dependencies that will lead us to the costume shape lattice. The benefit of this kind of lattice is that the dimension of the main object will not be affected by the other modifiers that try to remesh it.

Translating Shapes of the Lattice Structure Using Blender

So far we have learned many ways to create lattice structures, but what we haven't learned about is how we can create these holes with the shape that we want and the places that we choose to be. In this section, we are going to create a tool to translate our desired shape to the specified vertex of an object and then we will Boolean the difference between the translated object from the main object. We use many of the functions that we used in Creating a tool in Blender to instantly translate an object to the point we show with the cursor article.



Python Scripts

So let's get started:

```
import  bpy
import bmesh
import math

#####################################################################
#####                Utility Functions
#####################################################################
class BOOLEAN_TYPE:
    UNION = 'UNION'
    DIFFERENCE = 'DIFFERENCE'
    INTERSECT = 'INTERSECT'

def make_boolean(obj1, obj2, boolean_type):
    if not obj1 or not obj2:
        return

    modifier = obj1.modifiers.new(name='booly', type='BOOLEAN')
    modifier.object = obj2
    modifier.operation = boolean_type

    res = bpy.ops.object.modifier_apply({"object": obj1}, apply_as=
'DATA', mod-ifier=modifier.name)

    assert "FINISHED" in res, "Error"

def solidify(obj,offset,thickness):

    bpy.ops.object.modifier_add(type='SOLIDIFY')
    bpy.context.object.modifiers["Solidify"].offset = offset
    bpy.context.object.modifiers["Solidify"].thickness = thickness
    bpy.ops.object.modifier_apply(apply_as='DATA', modifier="Solidify"
)
```

The above function will convert a solid object to a shell of its shape.

```
def object_closest_point_mesh(p, obj):

    result, location, normal, face_index = obj.closest_point_on_mesh(p)
    assert result, "Can't find closest point on mesh"
    location = location.to_tuple()
    normal = normal.to_tuple()
    return location + normal  # return tuple of 6 floats

def obj_transform(filename, obj_name, size, location, angle):

    ob = bpy.context.scene.objects[obj_name]       # Get the object
    bpy.ops.object.select_all(action='DESELECT'
) # Deselect all objects
    bpy.context.view_layer.objects.active = ob   # Make the cube the active o
    ob.select_set(True)

    obj = bpy.data.objects[obj_name]
    obj.location = location

    bpy.ops.transform.rotate(value=angle, orient_axis='Z',
        orient_type='GLOBAL',
        orient_matrix=((1, 0, 0), (0, 1, 0), (0, 0, 1)),
        constraint_axis=(False, False, True))

def object_put_part2(part_name, point, obj, scale, obj_name):
    vx,vy,vz,a,b,c = object_closest_point_mesh(point, obj)
    a1 = math.atan2(b, a)
    obj_transform(part_name, obj_name, scale, (point[0], point[1
], point[2]), a1)

def get_vertex():
    bm = bmesh.new()
    ob = bpy.context.active_object
    bm = bmesh.from_edit_mesh(ob.data)

    points = []
    for v in bm.verts:
        if (v.select == True):
            obMat = ob.matrix_world
            points.append(obMat @ v.co)

    for p in points:
        pOb = bpy.data.objects.new("VertexPoint", None)
        bpy.context.collection.objects.link(pOb)
        pOb.location = p
    return p

def delete_object(objName):

    bpy.ops.object.select_all(action='DESELECT')
    bpy.data.objects[objName].select_set(True) # Blender 2.8x
    bpy.ops.object.delete()

def get_object_by_name(obj_name):

    assert obj_name in bpy.data.objects,
"Error getting object by name:{}".format(obj_name)
    obj = bpy.data.objects[obj_name]
    return obj

####################################################################
########             Main Panel
####################################################################

class MainPanel(bpy.types.Panel):
    bl_label = "Object Adder"
    bl_idname = "VIEW_PT_MainPanel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = 'Design Automation'

    def draw(self, context):
        layout = self.layout
        layout.scale_y = 1.2

        row = layout.row()
        row.label(text= "Design Automation", icon= 'OBJECT_ORIGIN')
        row = layout.row()
        row.operator("wm_function.myop", text= "Create single Lattice"
)
```

```
####################################################################
####                  Main UI ?Functions
####################################################################

class WM_Function_myOp(bpy.types.Operator):
    """Go to edit mode and determine the point then Click the button"
""
    bl_label = "Our customized function"
    bl_idname = "wm_function.myop"

    scale = bpy.props.FloatProperty(name= "Enter the scale of lattice"
, default= 1)
    Lattice_Name = bpy.props.StringProperty(name=
"Enter the scale of lattice", default= '')

    def execute(self, context):

        Scale = self.scale
        Lattice_Name = self.Lattice_Name
        point = get_vertex()
        bpy.ops.object.editmode_toggle()
        obj = get_object_by_name('Icosphere')
        solidify(obj, 0, 0.02)
        obj2 = get_object_by_name('%s'%Lattice_Name)
        object_put_part2('%s'%Lattice_Name, point, obj, Scale, '%s'
%Lattice_Name)
        make_boolean(obj,obj2,'DIFFERENCE')
        delete_object("VertexPoint")
        return {'FINISHED'}

    def invoke(self, context, event):

        return context.window_manager.invoke_props_dialog(self)
```

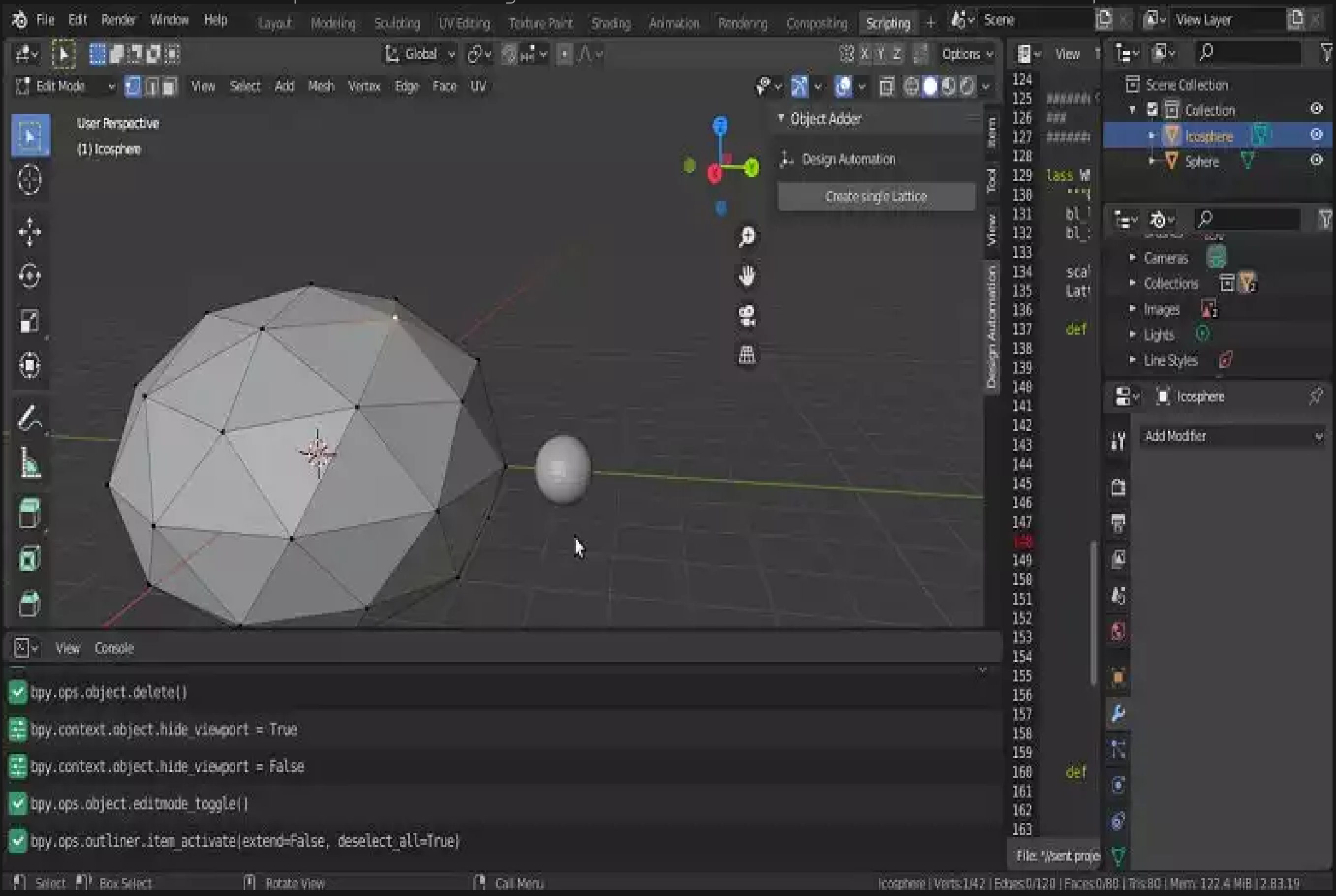
What We Have Done in the Scripts Above

In the above def execute function, at first get the vertex of the object, then we will create the shell of the main object. After that, we get the name of the lattice shape that has been imported or created by the user. In the next step, we will translate the lattice to the specified point by the user and finally we apply the boolean difference function to the objects. Do not forget the below scripts related to registering and unregistering the classes.

```
####################################################################
#####                        Register and Unregister
####################################################################

def register():
    bpy.utils.register_class(MainPanel)
    bpy.utils.register_class(WM_Function_myOp)

def unregister():
    bpy.utils.unregister_class(MainPanel)
    bpy.utils.unregister_class(WM_Function_myOp)

if __name__ == "__main__":
    register()
```

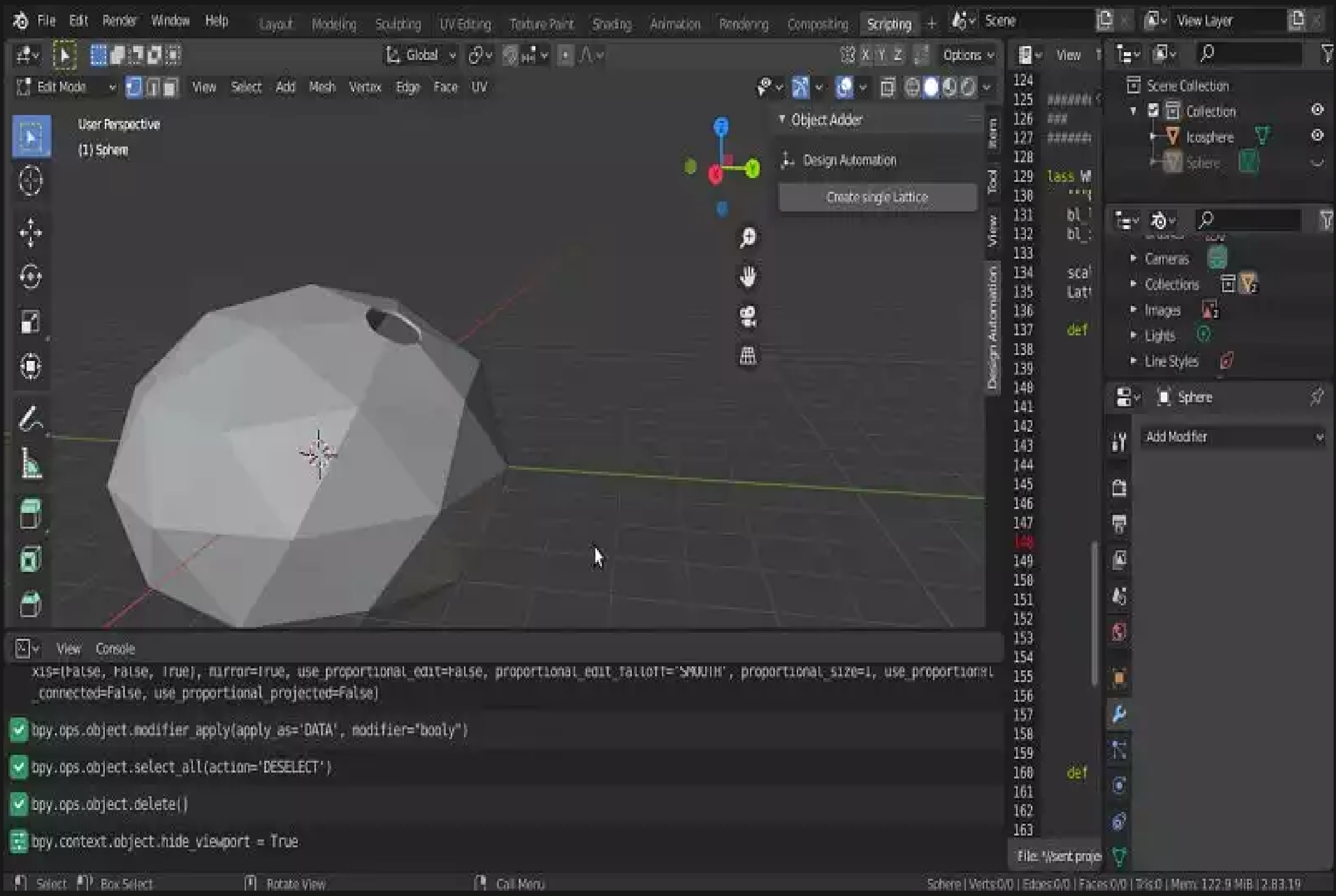
How to Use the Translating Tool

Now, it is time to test the panel we have designed and created. To do so, select a vertex on the object in edit mode:



```
bpy.ops.object.delete()
bpy.context.object.hide_viewport = True
bpy.context.object.hide_viewport = False
bpy.ops.object.editmode_toggle()
bpy.ops.outliner.item_activate(extend=False, deselect_all=True)
```

Then, simply click on the button on the panel (Create single lattice).



```
xis=(False, False, True), mirror=True, use_proportional_edit=False, proportional_edit_falloff='SMOOTH', proportional_size=1, use_proportional
 _connected=False, use_proportional_projected=False)
bpy.ops.object.modifier_apply(apply_as='DATA', modifier="booly")
bpy.ops.object.select_all(action='DESELECT')
bpy.ops.object.delete()
bpy.context.object.hide_viewport = True
```

And here we go! You can now see the lattice structure with the exact shape that you want on the object.

Wrapping Up

In this tutorial, we have managed to design an aesthetic internal lattice structure manually without the need for any coding python scripts. The design process is easy, however, you need to repeat and practice it in order to easily deal with the tricky parts.

Then, we got familiar with the tricks of creating aesthetic internal lattice structures and how to mesh formations such as triangular, voxel-based, and other ones that can influence the beauty of the design of these objects.

Moreover, we have managed to design a panel using python scripts (Blender Python API) to create any number of lattices with any shape and any size on our object. The only thing that the user interface wants from the user is the vertex on which the lattice is going to be placed. We can use this panel to create our custom shape lattice structure with accurate dimensions.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.

Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP NEWSLETTER RSS FEED