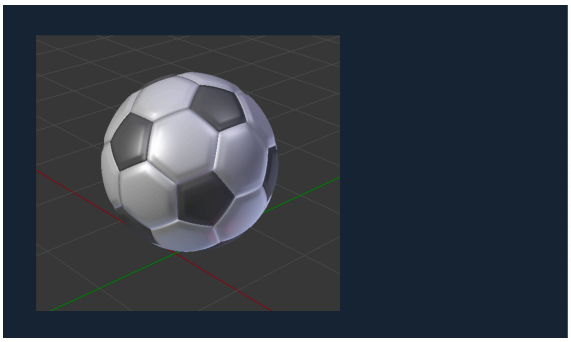
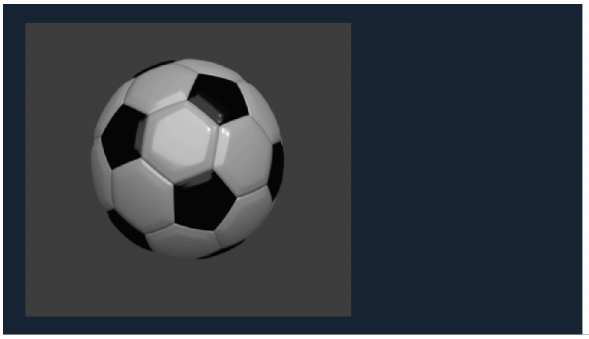
4.2 The Soccer Ball



soccer	1.000
football	1.000
ball	0.999
leather	0.998
gameplan (sports)	0.994
hexagon	0.994
game	0.994
pentagon	0.992
goal	0.991

(a)



soccer	1.000
football	1.000
leather	1.000
ball	1.000
game	1.000
gameplan (sports)	0.999
goal	0.999
sport	0.999

(b)

Figure 7. 3D models and Clarifai API predictions for a teddy bear (a) before rendering & (b) after rendering.

The second 3D model presented in this work is a soccer ball[1]. In last section, for the cup, the cylinder and the Bézier curve were already chosen at the beginning, and the color and rendering (white ceramic) was also determined in advance. Although the rendering was also selected for the soccer ball, the shape and the colors were not. Thus, the goal here is twofold: picking the right polygon, and choosing the correct color for each face on the chosen polygon.

One after the other, thirty different polygons were tried and, for each of them, nine pre-selected colors were applied on each of their faces. Due to the limitation on the number of predictions by the Google Vision API, only two colors were tested at once – that is, a polygon was painted with only two colors chosen from a set of nine, for a total of 9 possible combinations. Due also to the same limitation, a color was chosen between the two, and two connected faces could not be of that color.

[1] For this model, the Clarifai API was chosen instead of the Google Vision API, since the latter does not recognize soccer ball. See appendix 7.X for examples of probability predictions on pictures of soccer ball.

In Figure 7, it is seen that the Deep3D Generator could create a 3D model for a soccer ball. But it also created all the models in Figure 8. Since it is possible to have soccer balls of different colors, it is, at this point, hard to tell if the Deep3D Generator had trouble with choosing colors or not.

At least, we have yet another proof that it can select the right shape of an object. In the next section, we will confirm that the DG struggles in picking the right colors – and the right rendering too.

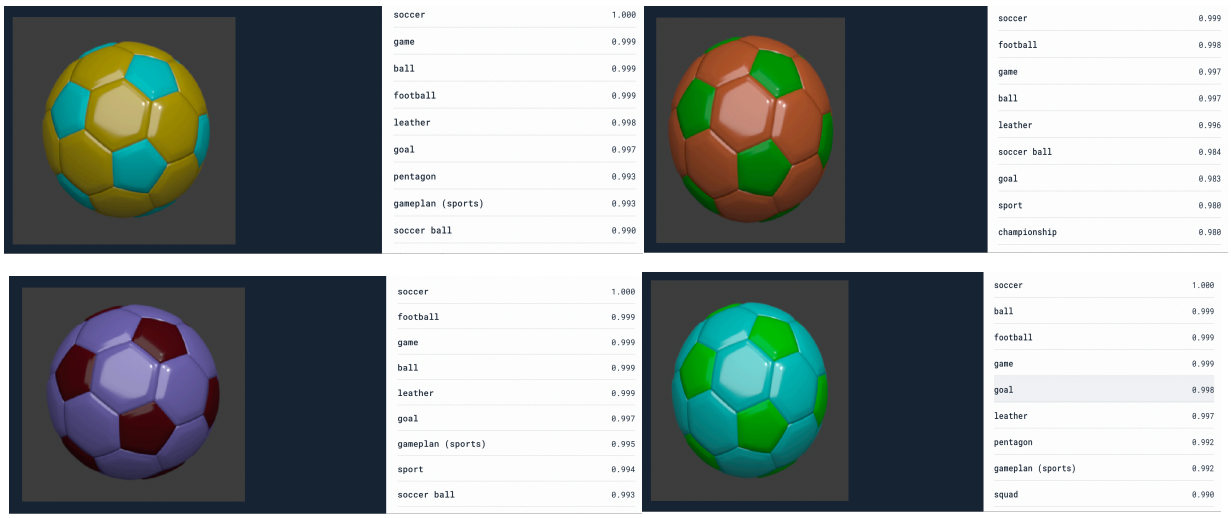
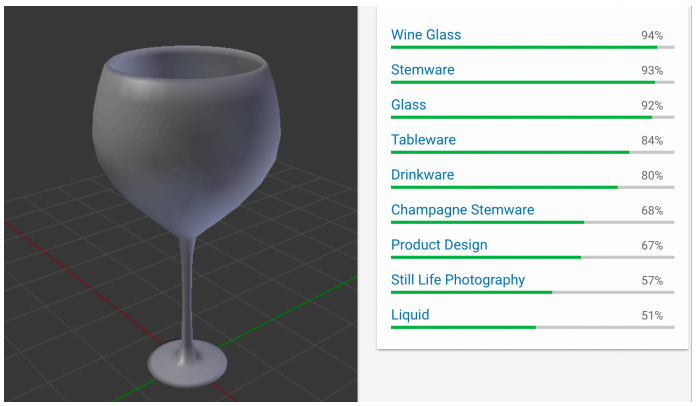


Figure 8. Other 3D models of a soccer ball accepted by the Deep3D Generator.

4.3 The Wine Glass

This third model was performed with the goal of experimenting with rendering. Since it has been shown in section 4.2 that the DG can be a little loose with the colors it admits in its 3D models, it is interesting to see if using rendering can improve the situation. A wine glass was chosen, since the material of a glass is complex enough to be unique, thus reducing the risk of mistakes by the Deep3D Generator; but simple enough to be rendered rather quickly.

The wine glass in Figure 9-a was obtained in a similar fashion as with the coffee cup, but starting only with a cylinder. The Deep3D Generator was set to stop for "Wine Glass" with probability of 93% with the rendering, and 90% without. Once the shape of the wine glass obtained, the DG is set to start exploring different rendering. The best result is shown in Figure 9-b, with the correct prediction of 96%.



(a)

(b)

Figure 9. 3D models and Google Vision API predictions for a glass of wine (a) before rendering & (b) after rendering. The orange plane in (b) was added afterwards (once the glass shape and the proper rendering was selected by the DG) to show the power of rendering, as it even produces the diffraction in the glass.

Although the results in Figure 9 are quite good, unfortunately, with a probability of 95%, the Google Vision API predicted the models in Figure 10 to be a wine glasses too. Those models still look like wine glasses due to their shape, but their applied rendering surely does not fit our expectations for real wine glasses. As it will be shown in section 4.4, this present model is not the only one for which the rendering could not be correctly chosen by the Deep3D Generator. We will be forced to conclude that this work, as it is currently presented here, will not be sufficient to generate models with their correct rendering – only the shape and colors. A solution to this problem will be presented in the discussion for future adjustments to the Deep3D Generator.

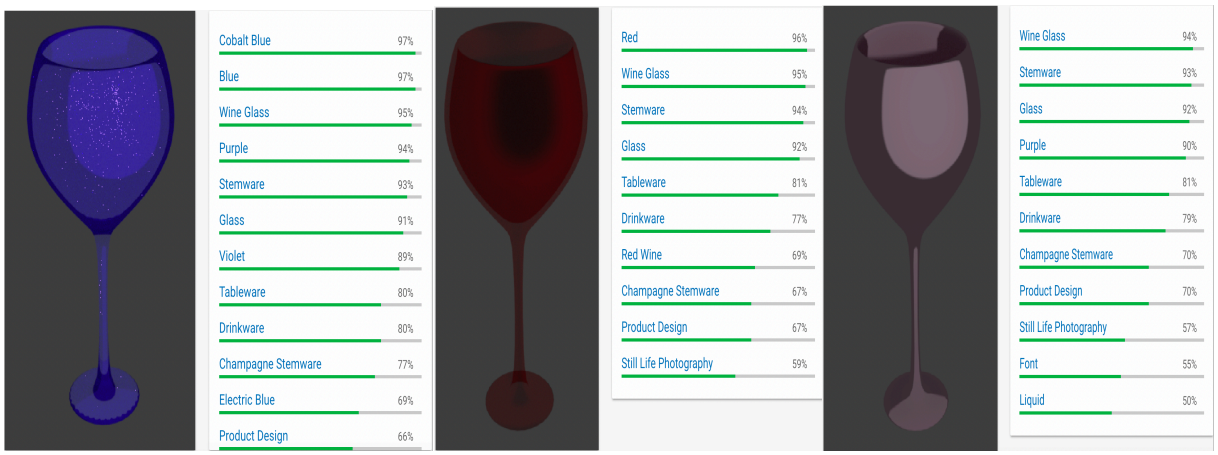


Figure 10. Other models for the wine glass, where rendering has been applied to the model in Figure 9-a. All the above have been retained by the Deep3D Generator.

4.4 The Teddy Bear

The teddy bear 3D model is composed of 15 spheres, all pre-selected before starting the Deep3D Generator. The interest of this 3D model was simply to make a 3D model more complex. All the spheres were programmed to touch each other to reduce the number of operations necessary to arrive to a teddy bear. As seen in Figure 11, it was possible to have two versions for the teddy bear: one with hair, the other without. For every model created as in Figure 11-a, it was possible to simply add the hair before taking the picture. The Deep3D Generator was thus testing two models at once. As it seen, both ways were successful.

Just like in the last section, however, the choice of colors was not successfully done by the DG. Since the 20 million limit did not allow much, only a few colors were tried on the model of Figure 11-b. But it was enough to show the problem. Although the 3D model in Figure 11-c is extraordinarily realistic, models in Figure 12 are not. Once again, the Deep3D Generator could select the right shape for the object, but not the right colors.

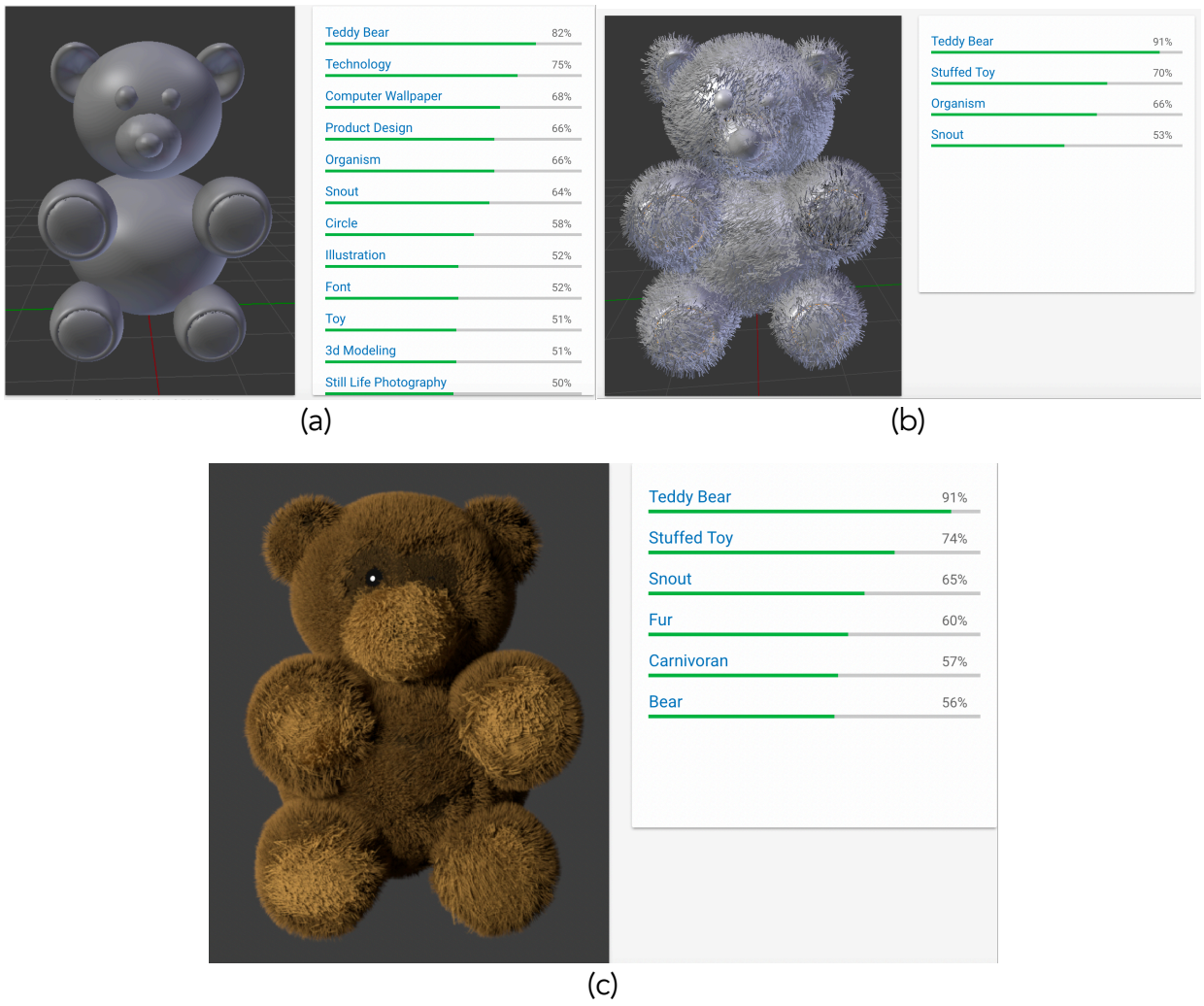


Figure 11. 3D models and Google Vision API predictions for a teddy bear with no hair added (a), and a teddy bear with hair added before rendering (b) and after rendering (c). For an unknown reason, the right eye is missing in (c).

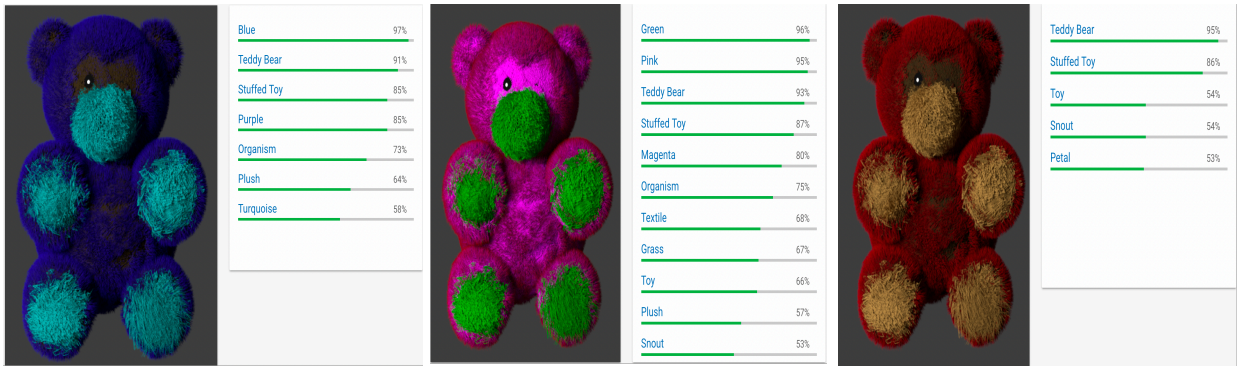


Figure 12. Other 3D models selected by the Deep3D Generator. They look like teddy bears, but their colors are not to be expected on a regular one.

5. Discussion

The Deep3D Generator could be improved in several ways. Although it is quite good to generate the correct shape of an object, it is obviously not for choosing the right colors and a proper rendering. This problem could be fix, maybe, as simply as in the following. To train the convolutional neural networks used in this work, one needs to have access to a huge database of images, with millions and millions of them. For each image in this database, it is possible the applied the newly trained CNN to get classification predictions, and create boundaries like in Figure 1. For a given object selected in these pictures, one can take the average color contained in the boundaries. One has thus the set of colors found in pictures for a given object. The Deep3D could use this set to choose the colors for its models. Since it is likely that, on average, all teddy bears are brown or maybe black, the 3D teddy bears generated by the DG will not be red or blue anymore – only brownish.

It would also be of interest to study and use symmetry in a more efficient way than it was done here. Group theory could be a good place to start. Having a better use of symmetry could reduce the number of operations needed to arrive to a successful 3D model – it would reduce the computing time and the required energy.

The 20 million limit on the number of classification predictions is also, obviously, a problem that would need to be fixed.

Once those above issues are fixed, the natural next step would be to create 3D models combining one or more objects together – i.e. a teddy bear with a hat, a cup of coffee with a spoon in it, etc. This would be a first step towards creativity. For an even more complex project, one could use the Deep3D Generator to create animations.

It would be also interesting to experiment with 3D printing.

6. Conclusion

In this work, it has been shown that artificial intelligence can be used to create 3D models. Although the Deep3D Generator presented here is not yet perfect, as it does not understand well how to pick colors and renderings for its models, it is believed that it can be improved to become a powerful tool with many applications.

7. References

[1] Yann LeCun, Yoshua Bengio, & Geoffrey Hinton, “Review: Deep Learning”, 27 May 2015, in Nature

[2] Shaoqing Ren, *et al*, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2015, arXiv:1506.01497

8. Appendix

8.1 Code for making the wine glass

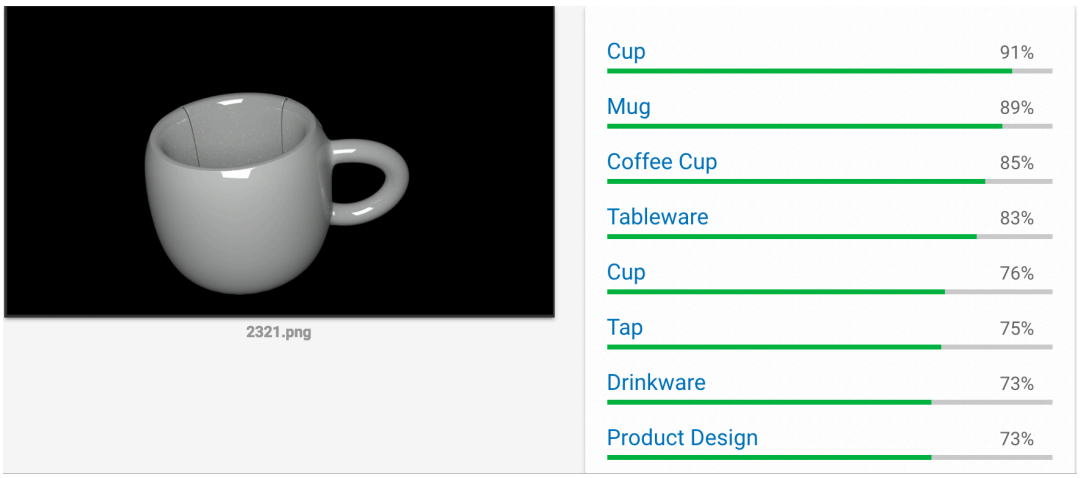
```
bpy.context.scene.render.engine = 'CYCLES'
ob = addEuclid.addCylinder(0,0,0)
bpy.ops.object.editmode_toggle()
bpy.ops.mesh.loopcut_slide(basicFunctions.overrideContext(), MESH_OT_loopcut={"number_cuts":3, "smoothness":0, "falloff":'INVERSE_SQUARE', "edge_index":81})
vertices=[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63]
doOperation.selectRing(ob,vertices)
doOperation.resize(0.861564, 0.861564, 0.861564,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.selectAll()
vertices=[65, 68, 71, 74, 77, 80, 83, 86, 89, 92, 95, 98, 101, 104, 107, 110, 113, 116, 119, 122, 125, 128, 131, 134, 137, 140, 143, 146, 149, 152, 155, 158]
doOperation.selectRing(ob,vertices)
doOperation.resize(1.09872, 1.09872, 1.09872,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.selectAll()
vertices=[66, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97, 100, 103, 106, 109, 112, 115, 118, 121, 124, 127, 130, 133, 136, 139, 142, 145, 148, 151, 154, 157]
doOperation.selectRing(ob, vertices)
doOperation.resize(1.05206, 1.05206, 1.05206,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.selectAll()
vertices=[64, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147, 150, 153, 156, 159]
doOperation.selectRing(ob, vertices)
doOperation.resize(0.745997, 0.745997, 0.745997,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.selectAll()
vertices=[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62]
doOperation.selectRing(ob, vertices)
doOperation.resize(0.177166, 0.177166, 0.177166,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.selectAll()
vertices=[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63]
doOperation.selectRing(ob, vertices)
doOperation.deleteThat('FACE')
doOperation.selectAll()
doOperation.enlargeThat(0,0,0)
doOperation.resize(1.07611, 1.07611, 1.07611,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.selectAll()
vertices=[160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222]
doOperation.selectRing(ob, vertices)
doOperation.enlargeThat(-1.77636e-15, -4.44089e-16, 0.154372)
doOperation.resize(0.590321, 0.590321, 0.590321,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
bpy.ops.transform.resize(value=(0.590321, 0.590321, 0.590321), constraint_axis=(False, False, False), constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED'
doOperation.enlargeThat(2.84217e-13, 1.7053e-13, 1.99458)
doOperation.selectAll()
vertices=[352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383]
doOperation.selectRing(ob, vertices)
doOperation.enlargeThat(-3.55271e-15, -7.10543e-15, 0.0891392)
doOperation.resize(4.16109, 4.16109, 4.16109,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.enlargeThat(-2.68674e-14, -3.26406e-14, 0.0516822)
doOperation.resize(2.19894, 2.19894, 2.19894,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.enlargeThat(3.44169e-14, 4.19664e-14, 0.0520294)
doOperation.resize(1.16067, 1.16067, 1.16067,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
```

8.2 Code for making the teddy bear in Figure 11

```
ob=addEuclid.addSphere(0,0,0)
doOperation.resize(1,1,0.85,False, False, False, 'GLOBAL', False, 'CONNECTED', 'SMOOTH', 1.0)
ob1=addEuclid.addSphere(0,0,2.5)
doOperation.changeEditMode()
vertices=[300]
doOperation.selectRing(ob1,vertices)
doOperation.translateSelect(0, 0, -0.3, False, False, True, 'GLOBAL', False, 'CONNECTED', 'SMOOTH', 2.1424)
doOperation.changeEditMode()
doOperation.resize(1.25,1.25,1.25, False, False, False, 'GLOBAL', False, 'CONNECTED', 'SMOOTH', 1.0)
ob2=addEuclid.addSphere(0,2.5,2.5)
doOperation.changeEditMode()
doOperation.selectAll()
vertices=[8, 9, 10, 11, 12, 13, 14, 23, 24, 25, 26, 27, 28, 29, 38, 39, 40, 41, 42, 43, 44, 53, 54, 55, 56, 57, 58,
59, 68, 69, 70, 71, 72, 73, 74, 83, 84, 85, 86, 87, 88, 89, 98, 99, 100, 101, 102, 103, 104, 113, 114, 115, 116, 117,
118, 119, 128, 129, 130, 131, 132, 133, 134, 143, 144, 145, 146, 147, 148, 149, 158, 159, 160, 161, 162, 163, 164, 173,
174, 175, 176, 177, 178, 179, 188, 189, 190, 191, 192, 193, 194, 203, 204, 205, 206, 207, 208, 209, 218, 219, 220, 221,
222, 223, 224, 233, 234, 235, 236, 237, 238, 239, 248, 249, 250, 251, 252, 253, 254, 263, 264, 265, 266, 267, 268, 269, 278,
279, 280, 281, 282, 283, 284, 293, 294, 295, 296, 297, 298, 299, 309, 310, 311, 312, 313, 314, 315, 324, 325, 326, 327, 328, 329,
330, 339, 340, 341, 342, 343, 344, 345, 354, 355, 356, 357, 358, 359, 360, 369, 370, 371, 372, 373, 374, 375, 384, 385, 386, 387, 388,
389, 390, 399, 400, 401, 402, 403, 404, 405, 414, 415, 416, 417, 418, 419, 420, 429, 430, 431, 432, 433, 434, 435, 444, 445, 446, 447, 448,
449, 450, 459, 460, 461, 462, 463, 464, 465, 474, 475, 476, 477, 478, 479, 480, 481]
doOperation.selectRing(ob2,vertices)
doOperation.translateSelect(0, 0, -2, False, False, True, 'GLOBAL', False, 'CONNECTED', 'SMOOTH', 2.1424)
doOperation.selectAll()
vertices=[0, 1, 15, 16, 30, 31, 45, 46, 60, 61, 75, 76, 90, 91, 105, 106, 120, 121, 135,
136, 150, 151, 165, 166, 180, 181, 195, 196, 210, 211, 225, 226, 240, 241, 255, 256, 270, 271, 285, 286, 300,
301, 302, 316, 317, 331, 332, 346, 347, 361, 362, 376, 377, 391, 392, 406, 407, 421, 422, 436, 437, 451, 452, 466, 467]
doOperation.selectRing(ob2,vertices)
doOperation.resize(1, 0.4, 1, False, True, False, 'GLOBAL', False, 'CONNECTED', 'SMOOTH', 1.0)
vertices=[481]
doOperation.selectRing(ob2,vertices)
doOperation.translateSelect(0,0,1,False,False,True,'GLOBAL', False, 'CONNECTED','SMOOTH',1)
doOperation.changeEditMode()
doOperation.resize(0.5,0.5,0.5,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.rotateSelected(-1.5708, 0,1,0, False,True,False, 'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.translateSelect(-0.143229, -0.177435, -2.41667, False, False, False, 'GLOBAL',False,'DISABLED','SMOOTH',1)
ob3=addEuclid.addSphere(0.89, 3.65, 0)
doOperation.resize(0.35,0.35,0.35,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.resize(0.3,1,1,True,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.translateSelect(0.24,0,0,True,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.translateSelect(0,-1.33,0,False,True,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
```

```
doOperation.translateSelect(0,0,0.097,False,False,True,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.changeEditMode()
doOperation.selectAll()
doOperation.changeEditMode()
doOperation.selectEntire("Sphere.003", False)
doOperation.selectEntire("Sphere", True)
bpy.ops.transform.translate(value=(0, 0, 4.11358), constraint_axis=(False, False, True),
constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, release_confirm=True)
doOperation.selectEntire("Sphere.001", True)
bpy.ops.transform.translate(value=(0, 0, -1.27), constraint_axis=(False, False, True),
constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, release_confirm=True)
doOperation.selectEntire("Sphere", False)
doOperation.selectEntire("Sphere.001", False)
doOperation.selectEntire("Sphere.002", True)
doOperation.selectEntire("Sphere.003", True)
doOperation.duplicateSelect(1, 1, 1)
doOperation.duplicateSelect(1, 1, 1)
doOperation.duplicateSelect(1, 1, 1)
doOperation.changeLocation("Sphere.002", 0.006, 1.009, 1.673)
doOperation.changeLocation("Sphere.003", 1.279, 1.007, 1.687)
doOperation.changeLocation("Sphere.004", 1.413, -0.884, 0.303)
doOperation.changeLocation("Sphere.005", 0.140, -0.881, 0.289)
doOperation.changeLocation("Sphere.006", 0.086, 0.799, 0.197)
doOperation.changeLocation("Sphere.007", 1.36, 0.796, 0.210)
doOperation.changeLocation("Sphere.008", 1.239, -0.927, 1.669)
doOperation.changeLocation("Sphere.009", -0.033, -0.924, 1.655)
doOperation.selectEntire("Sphere.008", False)
doOperation.selectEntire("Sphere.009", False)
doOperation.selectEntire("Sphere", True)
doOperation.duplicateSelect(1, 1, 1)
doOperation.resize(0.4,0.4,0.4,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.changeLocation("Sphere.010", 0.843, -0.039, 2.582)
doOperation.duplicateSelect(1, 1, 1)
doOperation.resize(0.4,0.4,0.4,False,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.changeLocation("Sphere.011", 1.168, -0.050, 2.578)
doOperation.selectEntire("Sphere.010", False)
ob4=addEuclid.addSphere(2, 2, 2)
doOperation.rotateSelected(-1.5708, 0,1,0, False,True,False, 'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.resize(0.5,1,1,True,False,False,'GLOBAL',False,'DISABLED','SMOOTH',1)
doOperation.changeEditMode()
doOperation.selectAll()
vertices=[481]
doOperation.selectRing(ob4,vertices)
doOperation.translateSelect(-0.5,0,0,True,False,False,'GLOBAL',False,'CONNECTED','SMOOTH',1)
vertices=[115]
doOperation.selectRing(ob4,vertices)
```

8.3 Picture of the coffee cup taken from above



It has been manually taken. This was not taken by the Deep3D Generator. This is just to show what the coffee cup looks like from above.