Volume Visualization

- Computer Graphics Project -

Project Report

[redacted]

Delft University of Technology
Computer Science and Engineering

Computer Science and Engineering
Delft University of Technology
https://www.tudelft.nl/

Title:
Volume Visualization

Project Group:
[redacted]

Participant(s):
Celal Karakoc, 4342933
[redacted]

Supervisor(s):
Prof. Dr. Elmar Eisemann

Copies: 1

Page Numbers: 24

Date of Completion:
November 9, 2018

Contents

Preface

Tasks	Celal	Ali	Irene
1a	33%	33%	33%
1b	33%	33%	33%
2	33%	33%	33%
3a	40%	30%	30%
3b	40%	30%	30%
4	40%	30%	30%
5	55%	30%	15%

Optional Tasks	Celal	Ali	Irene
1	35%	50%	15%
2a	100%		
2b	40%	20%	40%
3a	90%	10%	
3b	100%		
3c	60%	10%	30%
3d	100%		

Extra functionality	Celal	Ali	Irene
Task 0	100%		
Python Scripts	100%		
Multithreading	50%	50%	
Dynamic changing of scene	100%		
Presentation		100%	
Report	33%	33%	33%

Delft University of Technology, November 9, 2018

Celal Karakoc
c.karakoc@student.tudelft.nl

[redacted]
[redacted]

[redacted]
[redacted]

Task 0 Data

Before starting to code we looked at the data itself. With this, we know for certain that the boundaries that we chose are the best that could have been chosen and it doesn't boil down to guess work. The python scripts to create such plots are found in Appendix **A**.

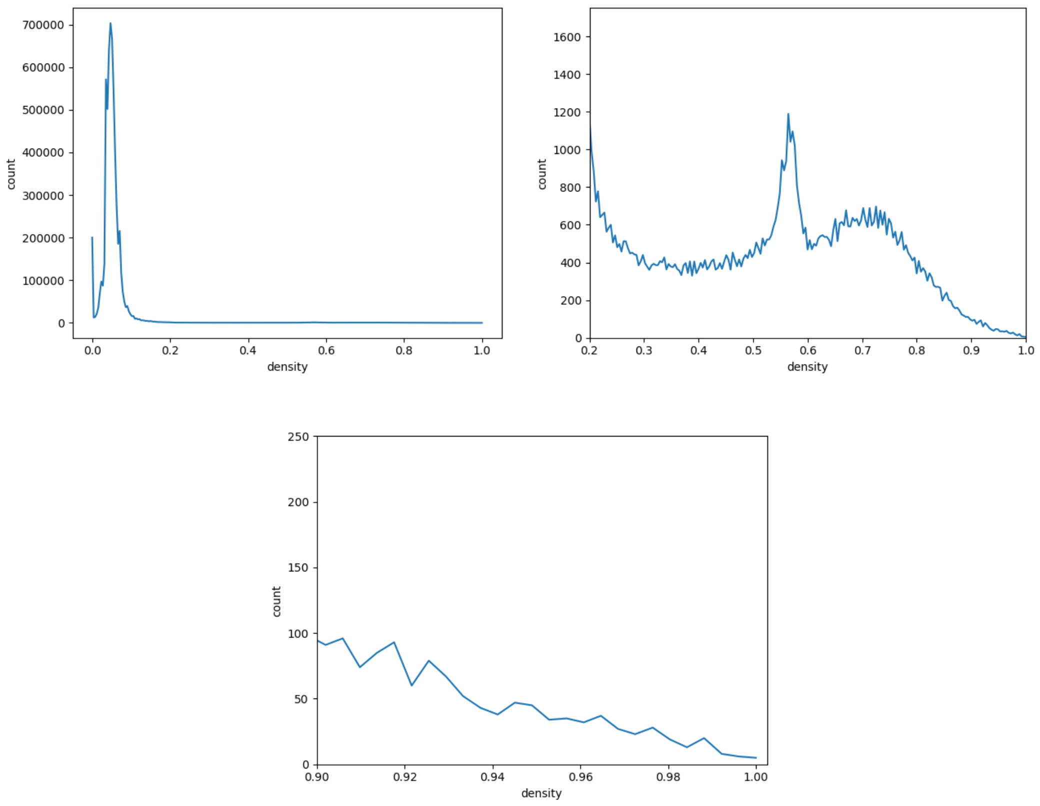

Figure 1: Bonsai and corresponding zooms

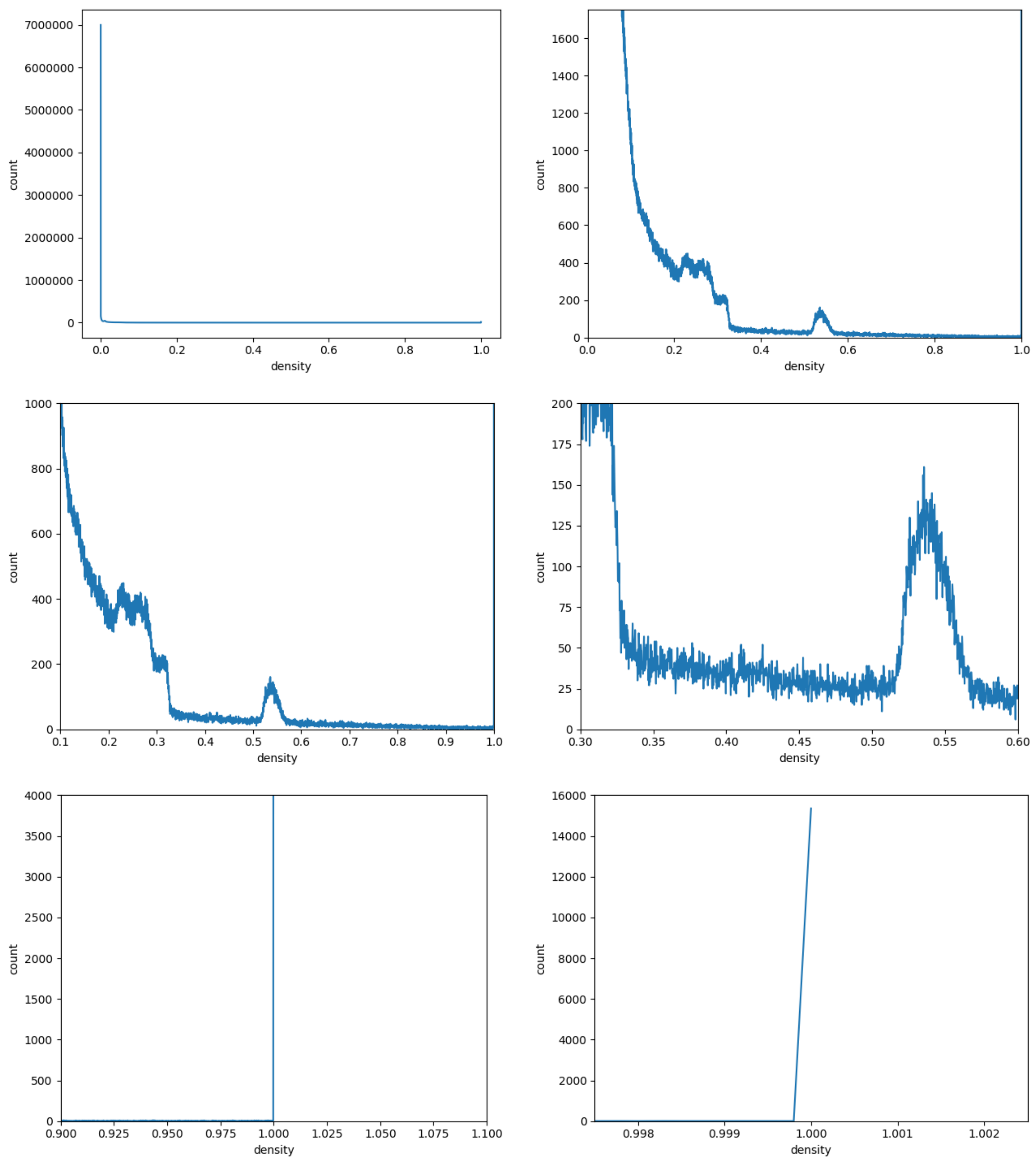

Figure 2: Backpack and corresponding zooms

Task 1 Point based visualization

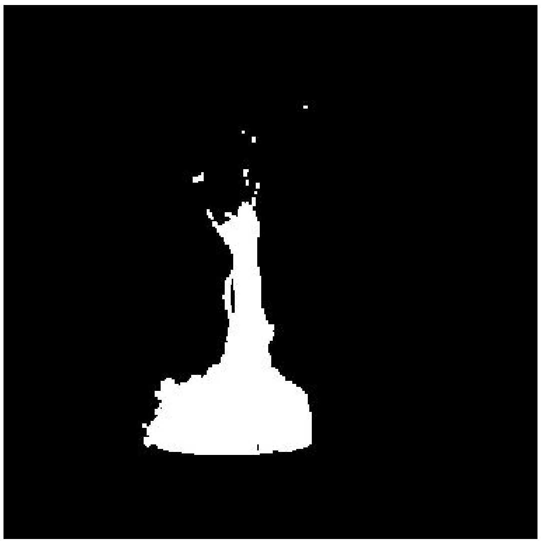
Figure 1.1: Bonsai with threshold=0.5 (task 1.a)

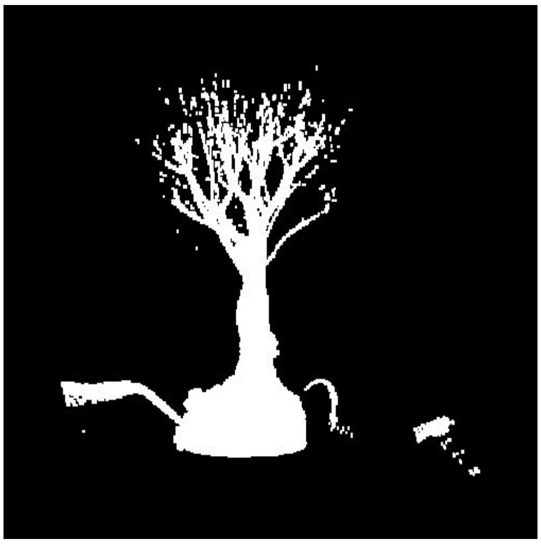
Figure 1.2: Bonsai between 0.2-0.6 (Task **0)**

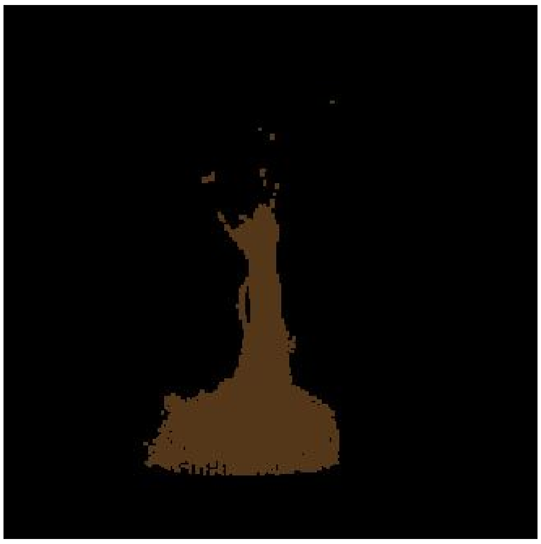
Figure 1.3: Bonsai between 0.5-0.6 (Task 1.b)

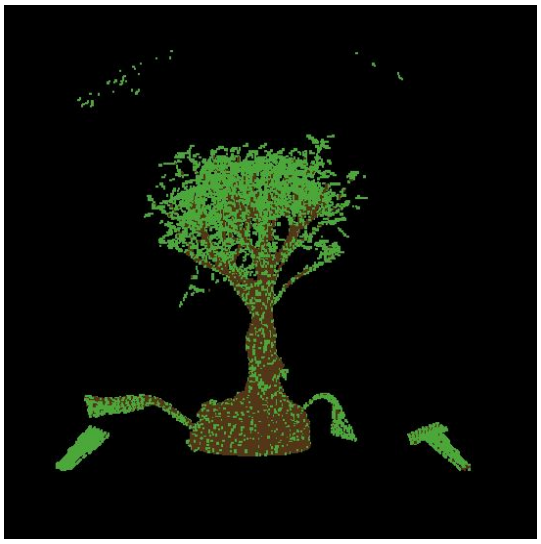
Figure 1.4: Bonsai with trunk and leaves (Task 1.b)

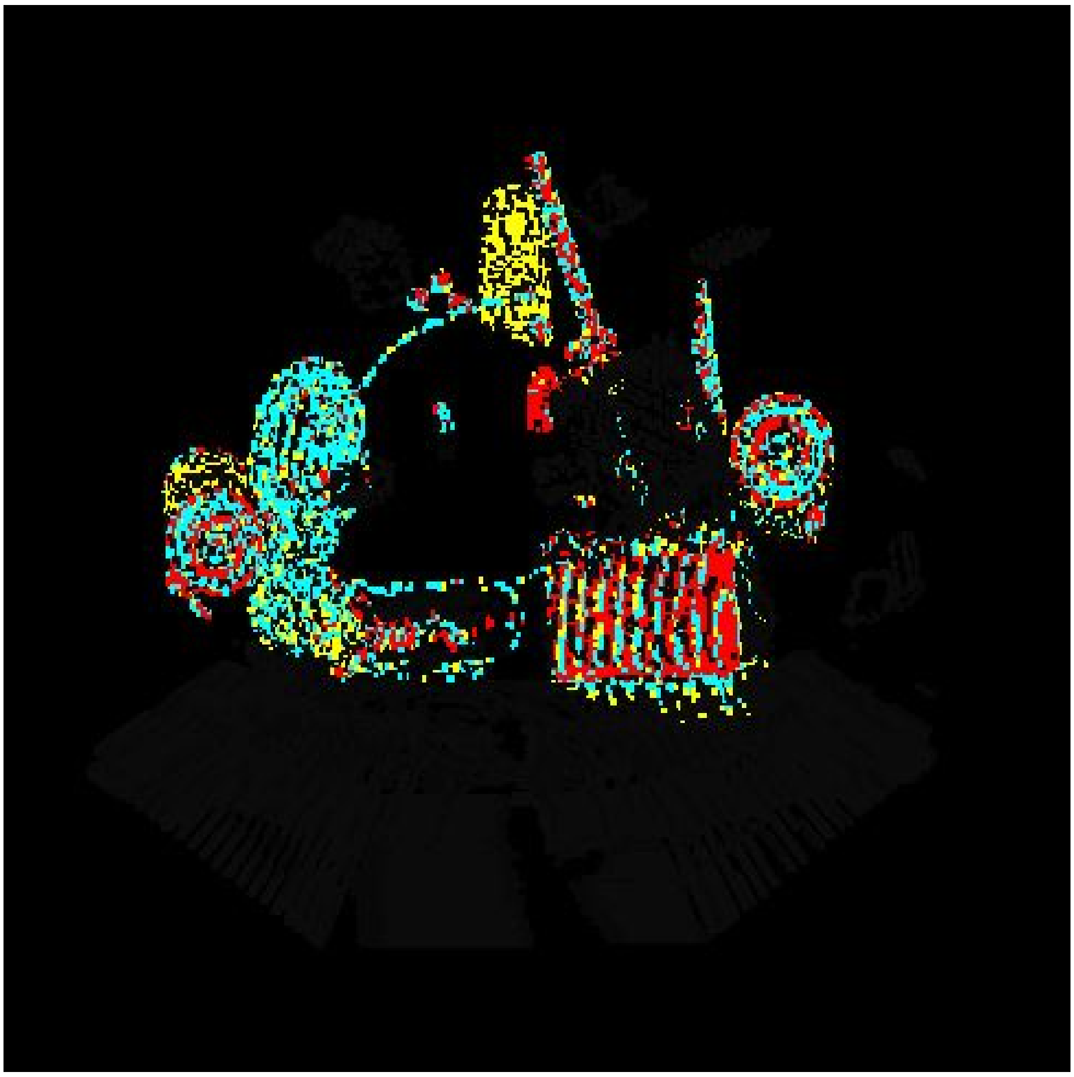
Figure 1.5: Backpack with Task 1 (Not actually part of the task)

Task 2 Blending

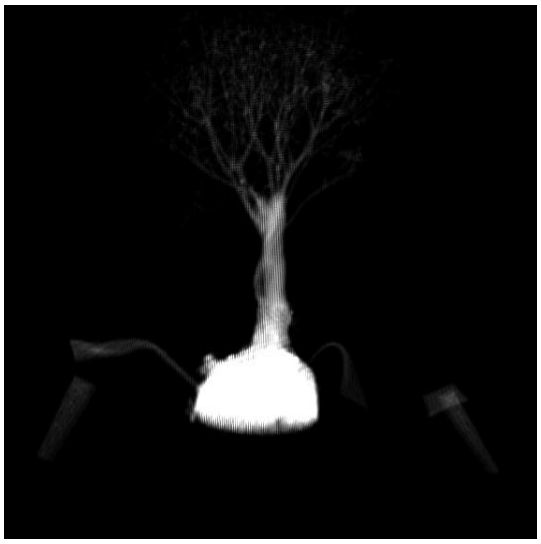

Figure 2.1: Bonsai monotone (0.05)

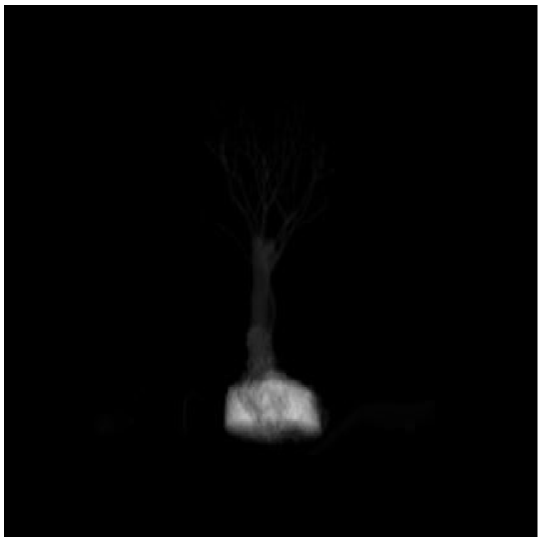

Figure 2.2: Bonsai monotone (0.005)

After the monotone of figure 2.1 and 2.2, we need to add color. We take the same brown and green for the trunk/leaves again. However, because of additive blending you will notice that the color is really intense (figure 2.3). So we make the trunk darker by multiplication (figure 2.4). Finally, the leaves could also become darker, with the final result being figure 2.5.

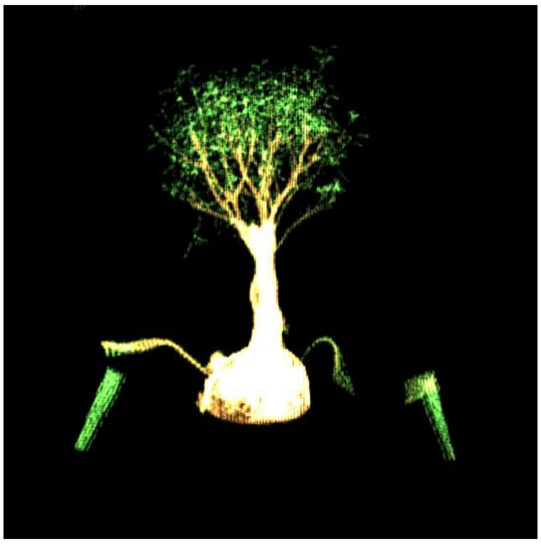
Figure 2.3: Bonsai color

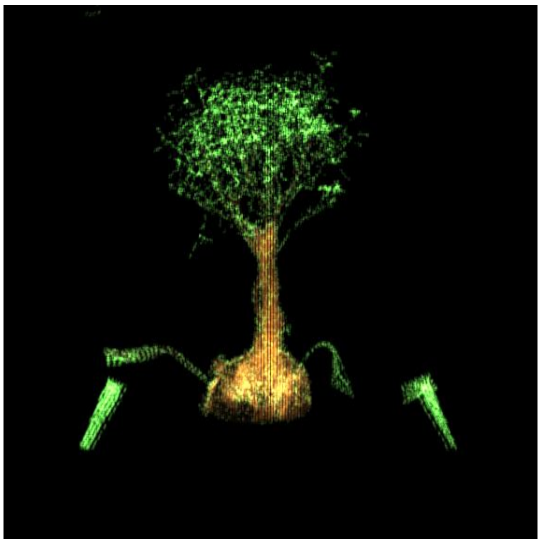
Figure 2.4: Bonsai color (darker trunk)

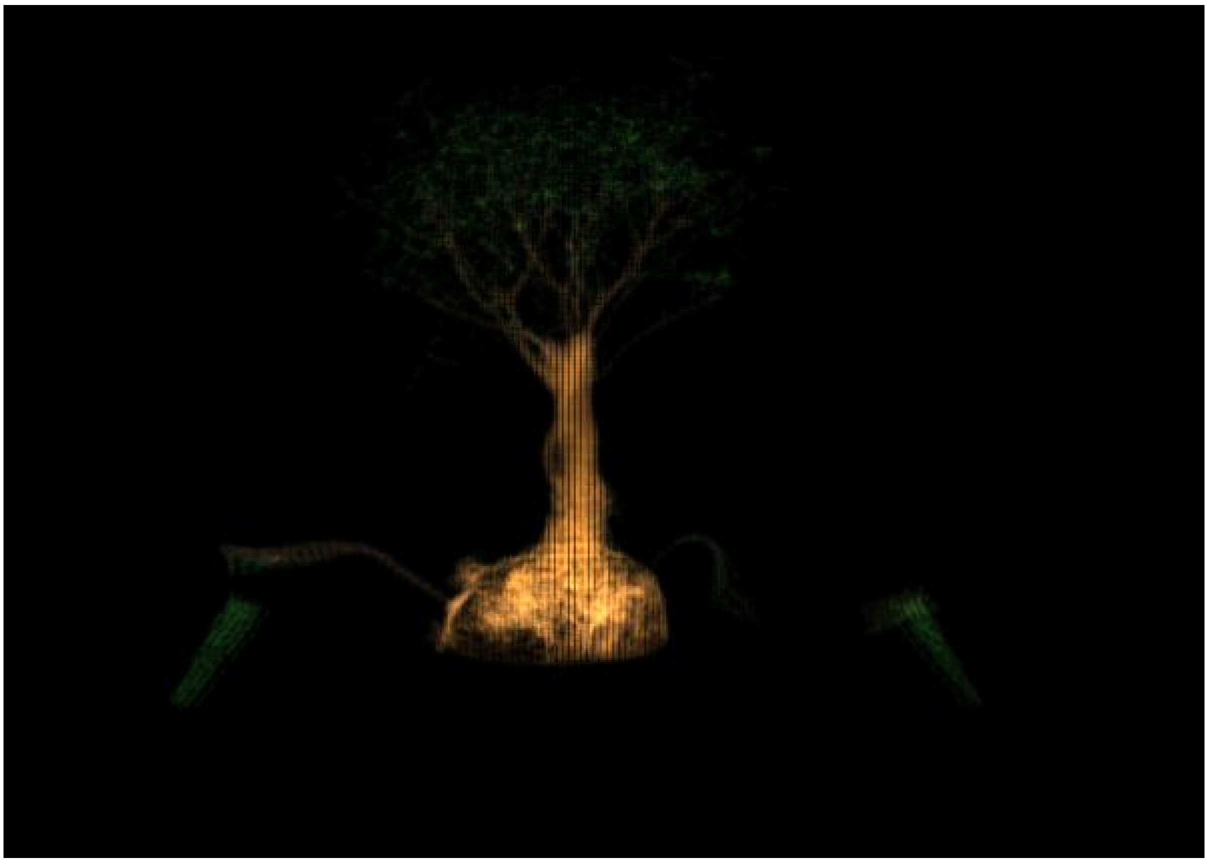
Figure 2.5: Bonsai color (darker)

Now we do the same process for the backpack, with the results being:

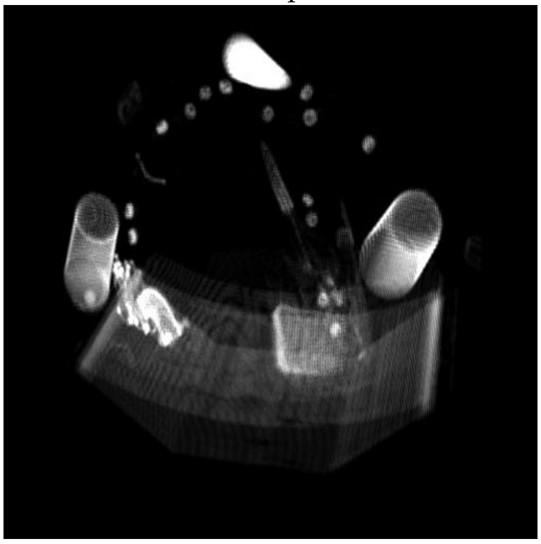

Figure 2.6: Backpack monotone (0.05 - up)

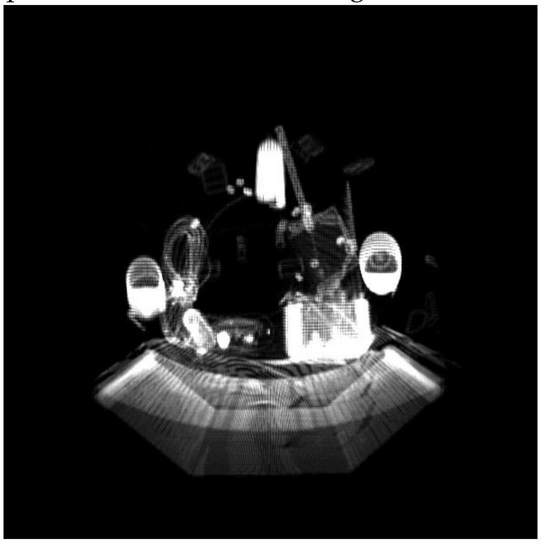

Figure 2.7: Backpack monotone (0.05 - front)

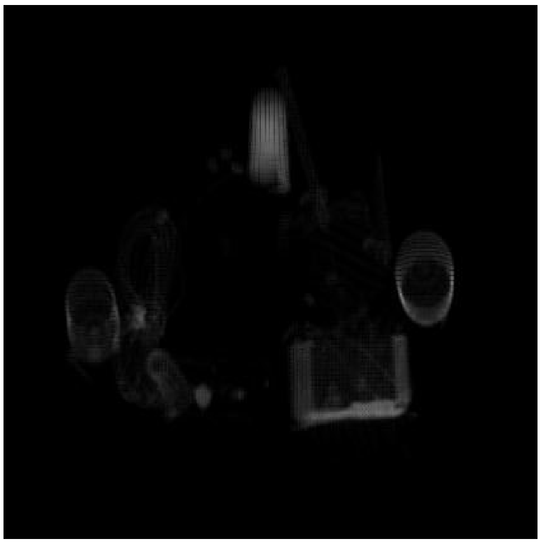

Figure 2.8: Backpack monotone (0.005)

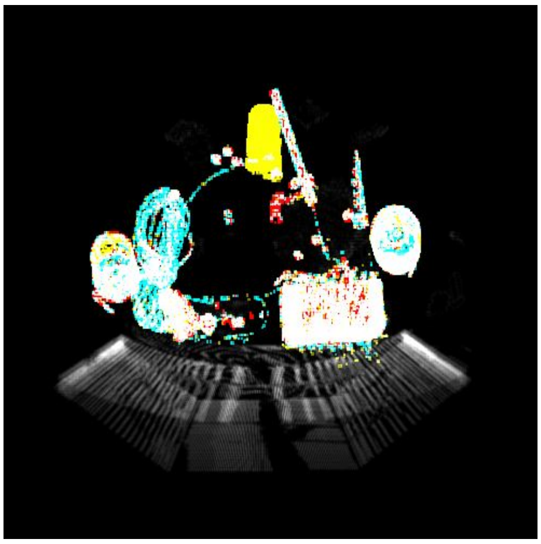

Figure 2.9: Backpack color

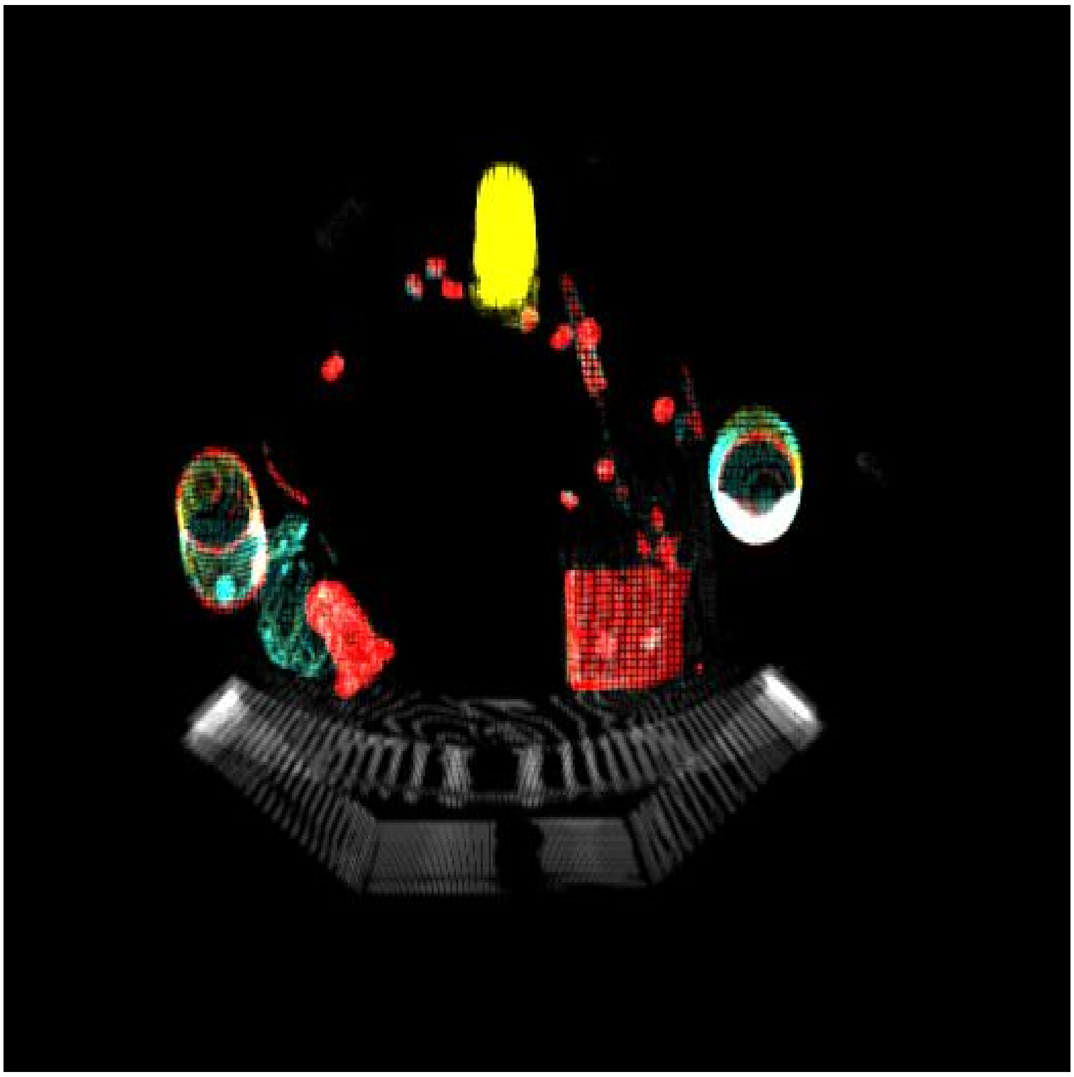

Figure 2.10: Backpack color (darker)

Task 3 Transfer functions and Accurate transparency

We adapted our code as such that we always draw back-to-front no matter our camera angle. This makes sure that no incorrect alpha blending is processed. We changed the boundaries based on the results of our data from Task 0. Furthermore, the formula that we use is: $\frac{1}{1+(\frac{\rho}{1-\rho})^{-Z}}$, where our Z is 1.5. We realized that with a higher density, you would need a higher alpha (so less transparency) and with a lower density a lower alpha (so more transparency). With this we searched for an S-shaped curve (like a sigmoid function) and we found this one. We tried several Z configurations, but 1.5 seemed to work best.

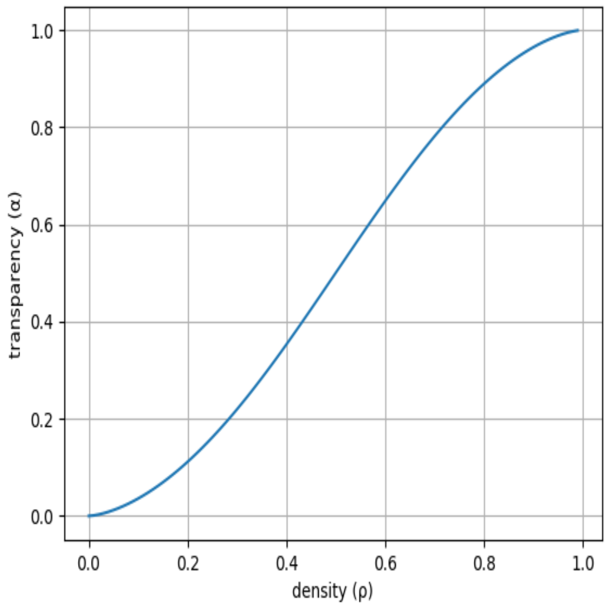

Figure 3.1: Mapping of ρ to α with Z=1.5

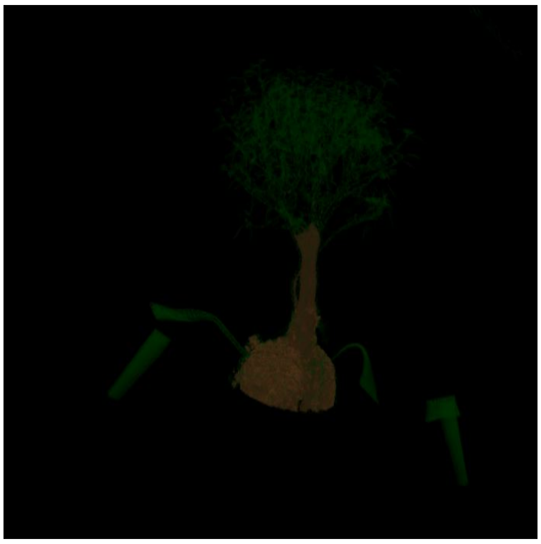
Figure 3.2: Bonsai (Task 3)

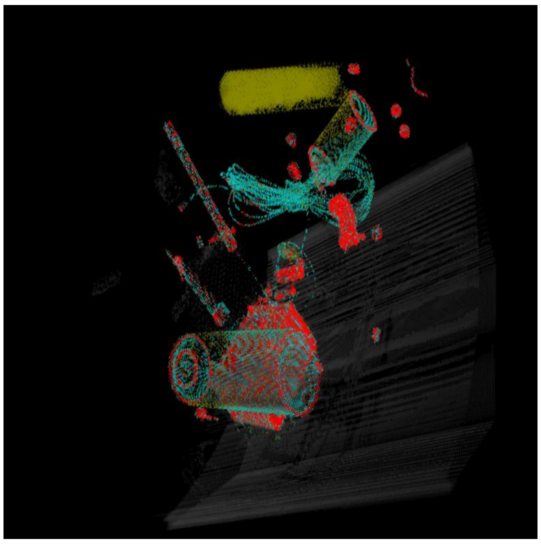
Figure 3.3: Backpack (zoom)

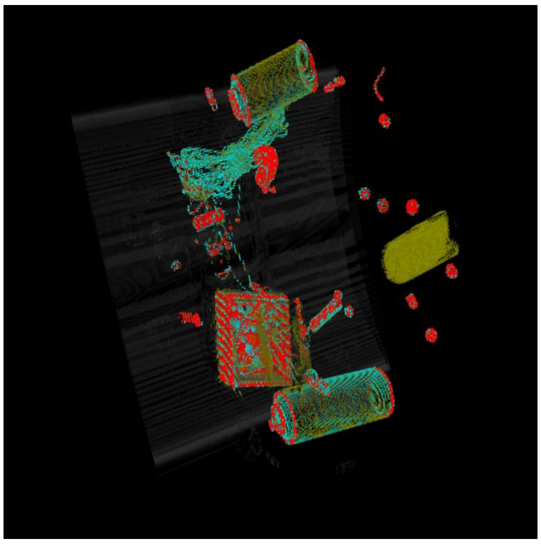
Figure 3.4: Backpack (above)

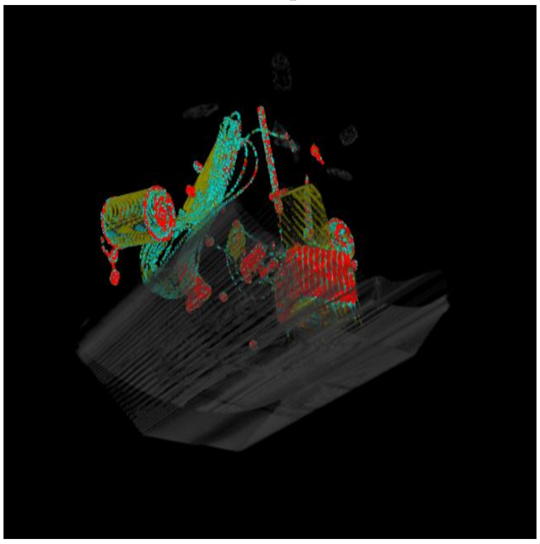
Figure 3.5: Backpack (below)

Task 4 Lighting

The light can be placed at the camera position, with l. The light can be moved with , . ; \ [ ]

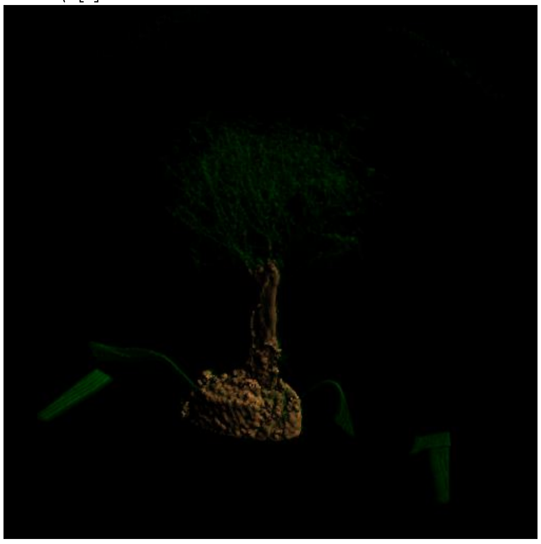
Figure 4.1: Bonsai

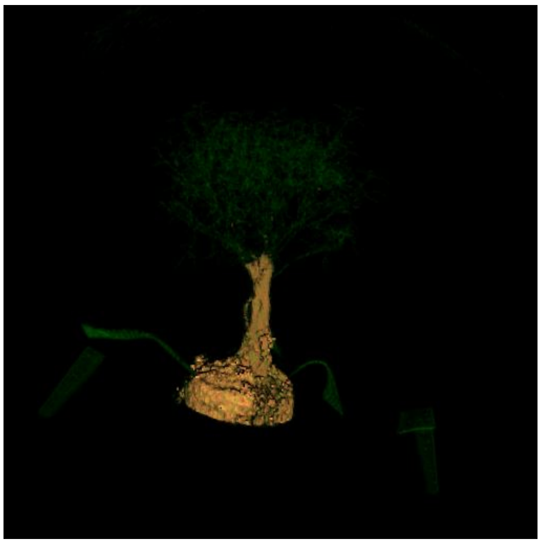
Figure 4.2: Bonsai (light from camera)

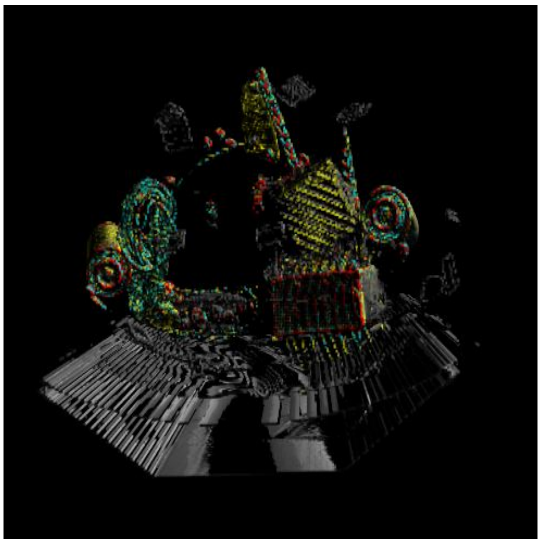

Figure 4.3: Backpack (1)

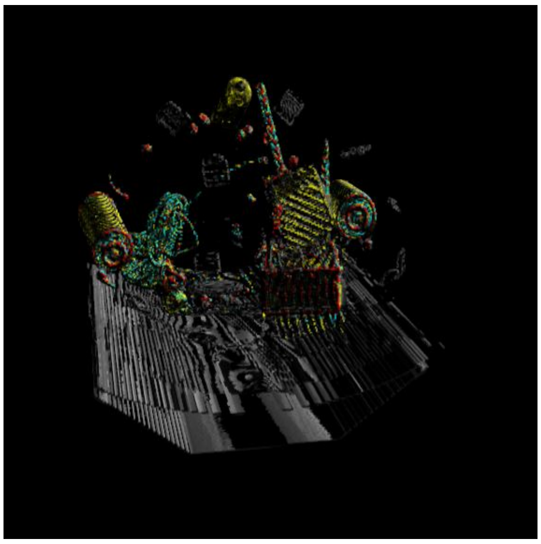

Figure 4.4: Backpack (2)

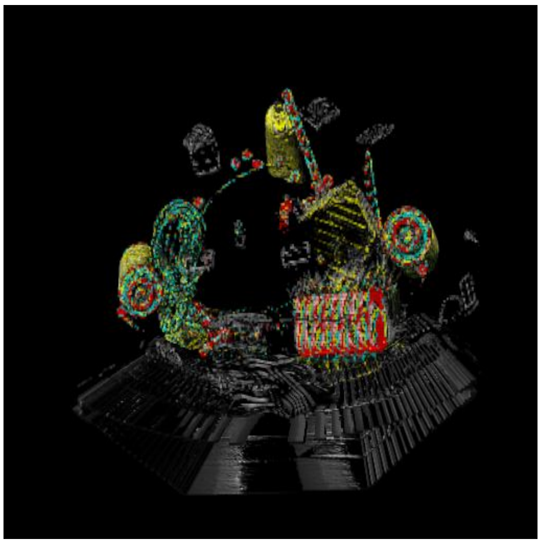

Figure 4.5: Backpack (light from camera)

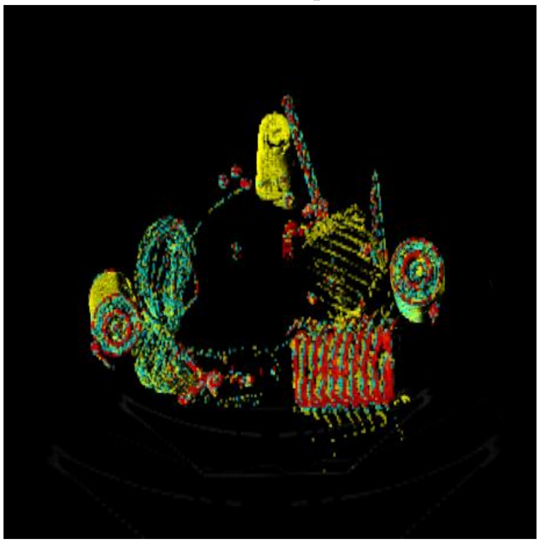

Figure 4.6: Backpack (light from camera + no grey)

Task 5 Intersection

All three shapes can be moved by the *w*, *a*, *s*, *d*, *e*, *q* keys. The sizes are changed with *x* and *z*. A note for the slab: the slab is drawn parallel and the plane is always as big as the entire object, so it can basically be seen as an infinite plane. You can move the axis on which the plane is drawn with *o*.

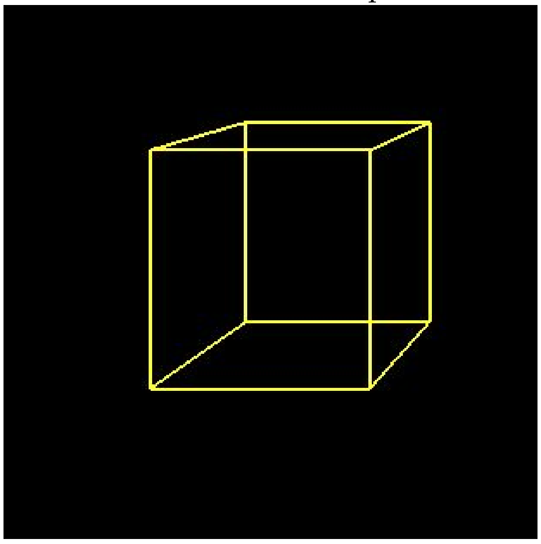

Figure 5.1: Cube (Task 5.a)

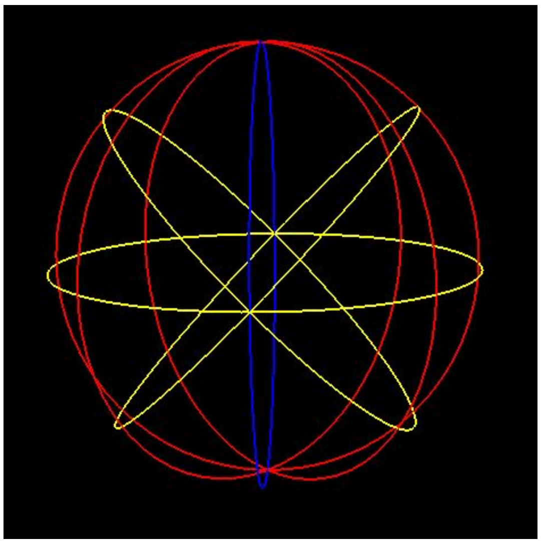

Figure 5.2: Sphere (Task 5.b)

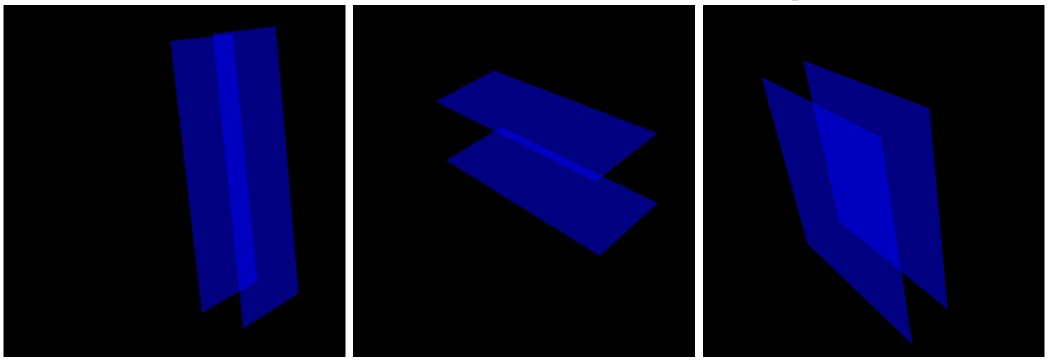

Figure 5.3: Slab (Task 5.c)

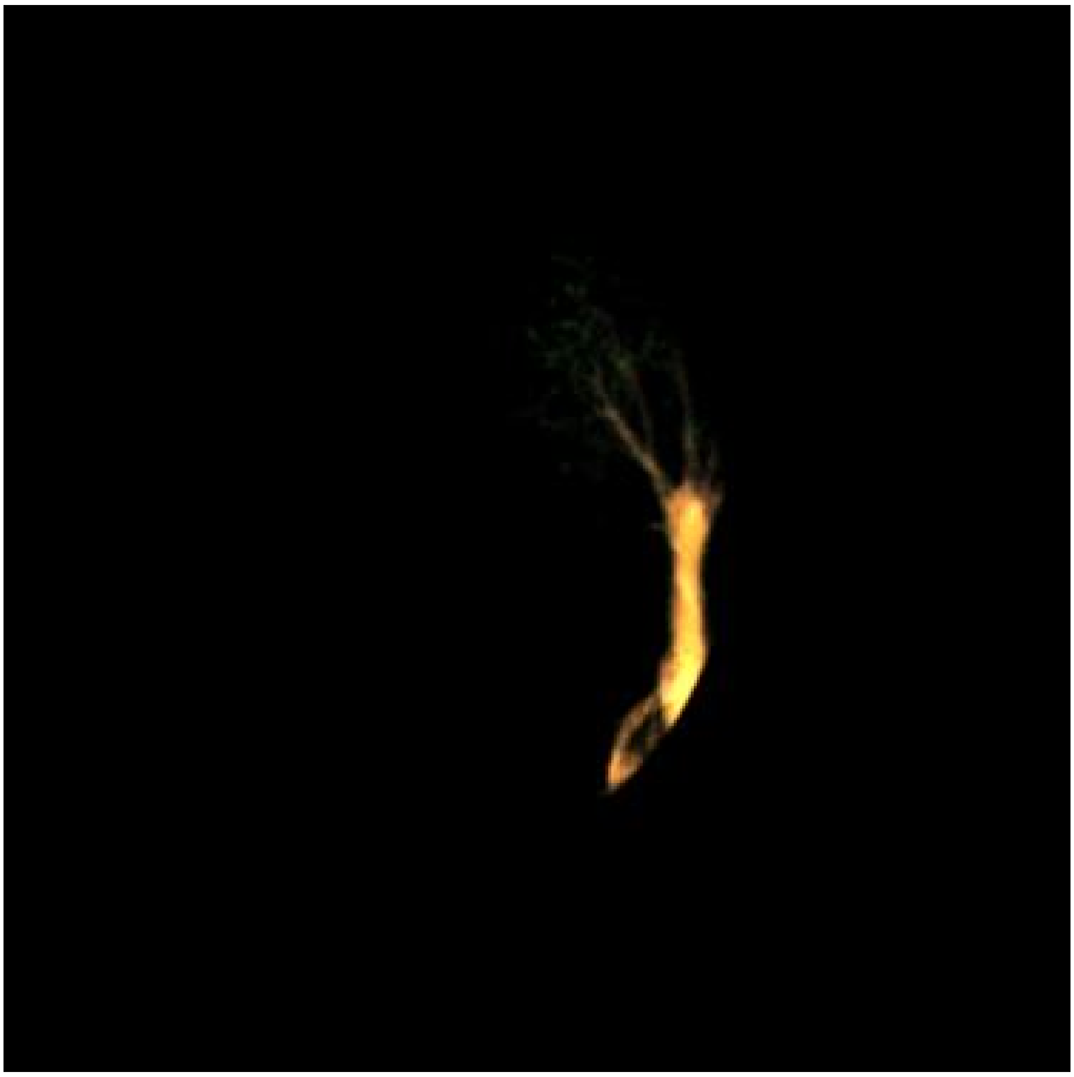

Figure 5.4: Example where the sphere is used to cut a part of the bonsai

Task 6 Trilinear interpolation

The trilinear interpolation is calculated based on the distance with the camera.

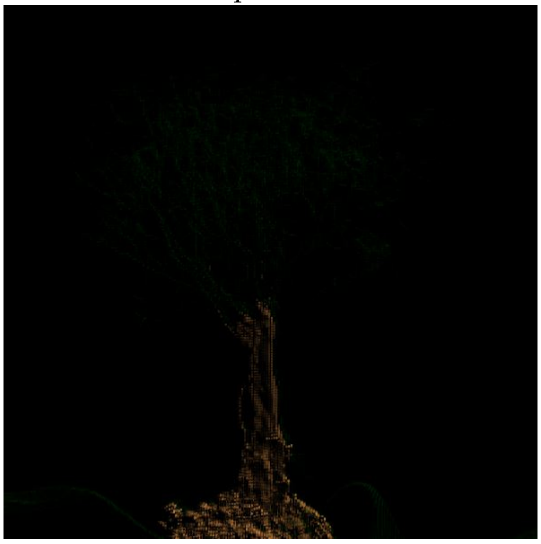
Figure 6.1: Bonsai

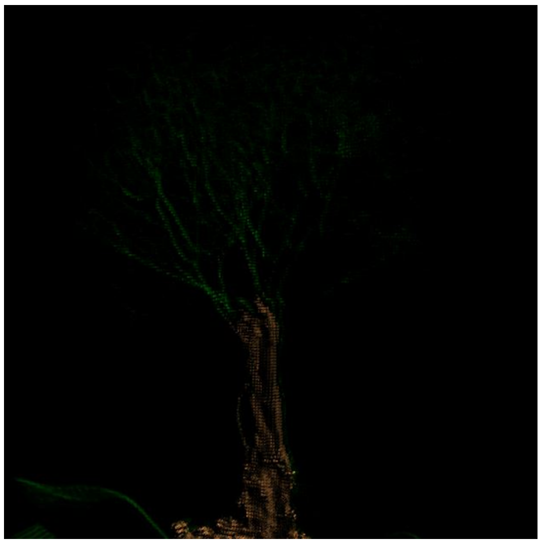
Figure 6.2: Bonsai (with trilinear interpolation)

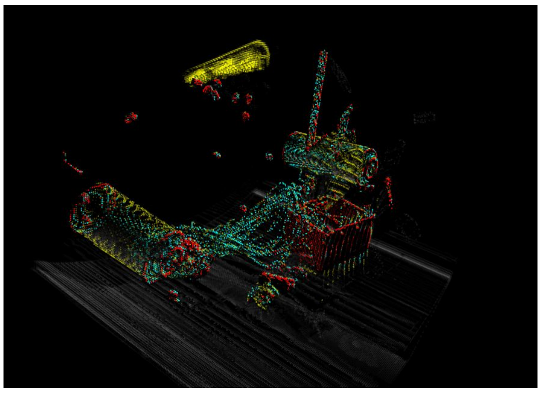
Figure 6.3: Backpack

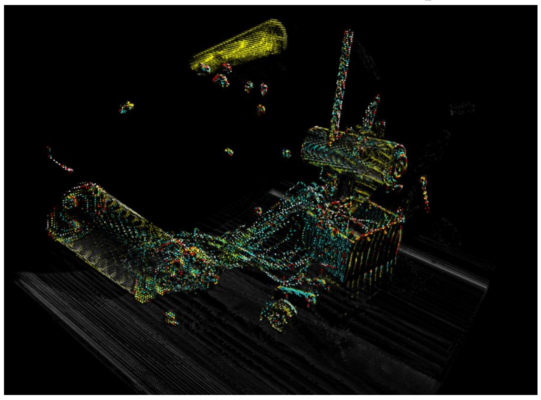
Figure 6.4: Backpack (with trilinear interpolation)

Task 7 Billboards

We use Quads for our billboards. While we could draw a circle with the triangle fan. The circle always facing the camera couldn't be implemented in time.

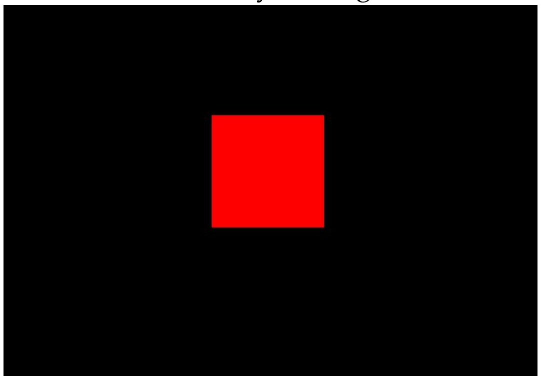
Figure 7.1: Quad

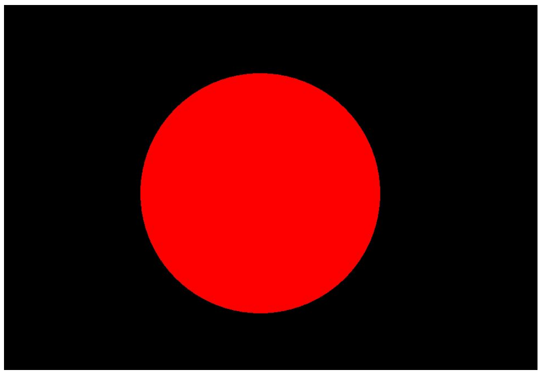
Figure 7.2: Triangle fan (90 vertices)

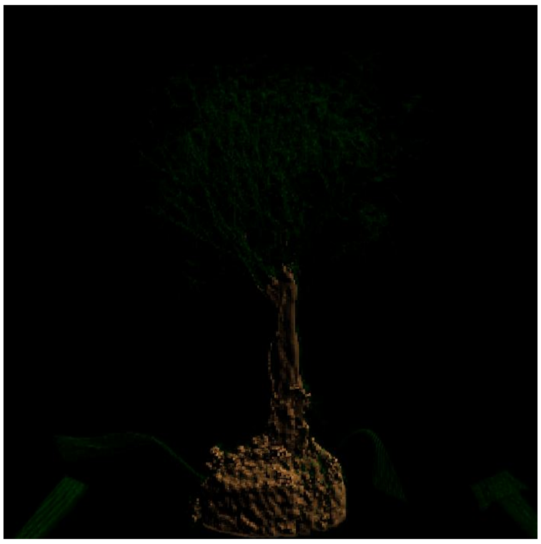
Figure 7.3: Bonsai (size is 0.005)

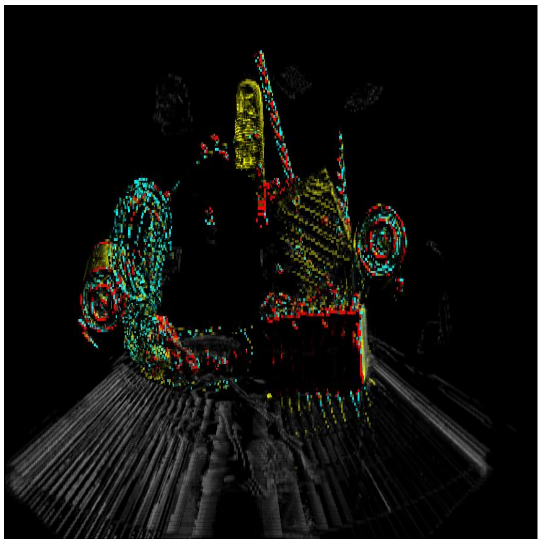
Figure 7.4: Backpack (size is 0.005)

Now let's have some fun with the size ⌣̈ (figure 7.5/7.6). The bonsai (7.5) doesn't look that bad, actually.

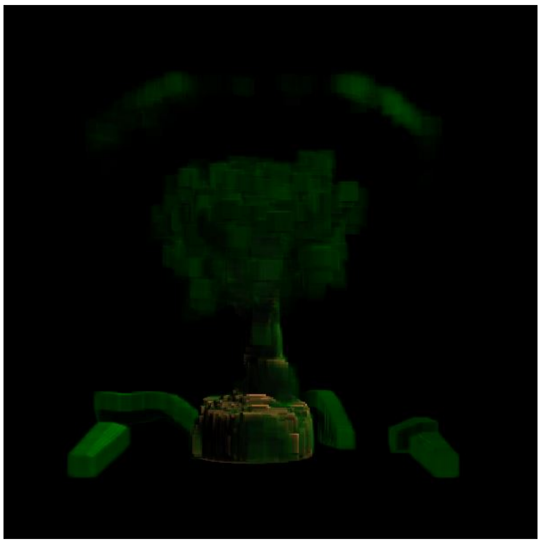
Figure 7.5: Bonsai (size is 0.5)

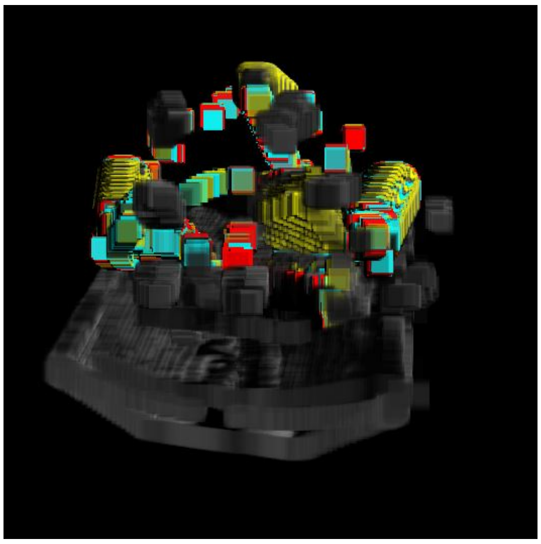
Figure 7.6: Backpack (size is 0.5)

For the LOD billboards: we have implemented a version, but there are slight problems with it. Basically the switching between big billboards and small billboards happens in a 'snap'. Besides that, all else is working.

Task 8 Efficient drawing

a

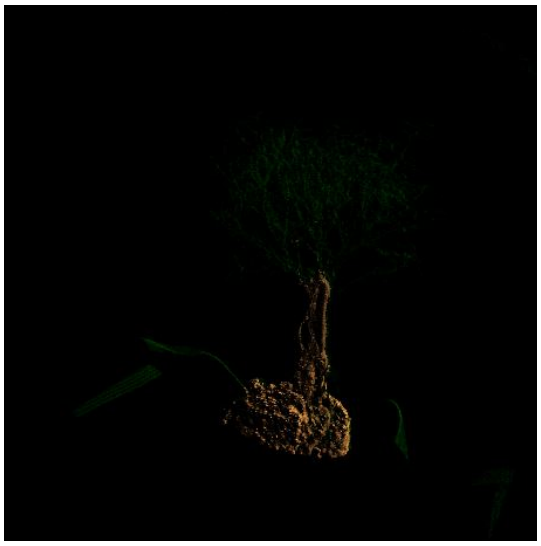

Figure 8.1: Bonsai (drawn with array)

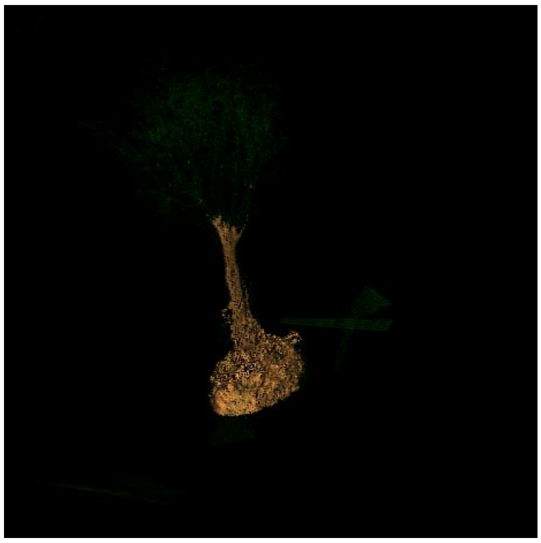

Figure 8.2: Bonsai (dynamically drawn when light changed)

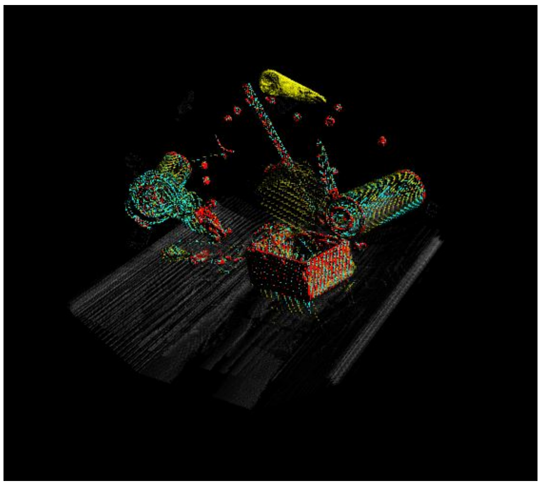
Figure 8.3: Backpack (drawn with array)

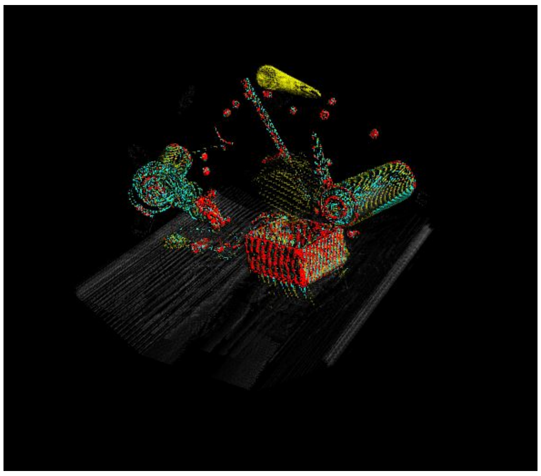
Figure 8.4: Backpack (dynamically drawn when light changed)

b

Optional task 3b was also done and produced the same results as 3a (Task 8.a in report), thus we didn't feel the need to provide a set of new images.

c

Optional task 3c was done a little bit different than described in the exercise. Before the exercise we noticed this very same problem and adapted our code to fix it. With that, we didn't really need to do anything else. As such, we didn't need to create several different vectors, because we fixed it dynamically.

d

	Bonsai (in ms)	Backpack (in ms)
Task 1	126.135	364.369
Task 2	133.19	362.306
Task 3	160.654	639.399
Task 4	314.869	935.548
Optional Task 1a	308.283	1064.06
Billboards	414.775	1192.99
Optional Task 3a	0.90288	2.43896
Optional Task 3b + 3c	0.84144	2.35272

Extra Functionality

- **Dynamic switching** Added dynamic switching between volumes (bonsai/backpack/other).
- **Multithreading** Added parallel loops.
- **Movement light** Added commands to move light in the scene.

Bibliography

[1] *Lighthouse billboarding*. URL: http://www.lighthouse3d.com/opengl/billboarding/index.php.

[2] *S-curved graph - Machine Learning*. URL: https://stats.stackexchange.com/questions/214877/is-there-a-formula-for-an-s-shaped-curve-with-domain-and-range-0-1.

[3] *Saving an Object (Data persistence)*. URL: https://stackoverflow.com/questions/4529815/saving-an-object-data-persistence.

[4] *Wikipedia Trilinear Interpolation*. URL: https://en.wikipedia.org/wiki/Trilinear_interpolation.

Appendix A Python Scripts

A.1 Save/Read density

```
import numpy as np
from collections import defaultdict
import pickle

def save_obj(obj, name):
        with open('obj/' + name + '.pkl', 'wb') as f:
                pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)

dimx, dimy, dimz = 256, 256, 91  # 91 becomes 181 for backpack
data = np.zeros((dimx, dimy, dimz))
with open('./in.txt') as f:
        lines = f.readlines()
        x, y, z = 0, 0, 0
        for line in lines:
                if '-' in line: # indicates a z-delimiter
                        x, y = 0, 0
                        z += 1
                else:
                        for val in line.split():
                                data[x, y, z] = val
                                x += 1
                x = 0
                    y += 1
res = defaultdict(int)
for x in range(dimx):
        for y in range(dimy):
                for z in range(dimz):
                        res[round(data[x, y, z], 4)] += 1
save_obj(res, 'out')
```

A.2 Load/Plot density

```
import matplotlib.pyplot as plt
import pickle

def load_obj(name):
        with open('obj/' + name + '.pkl', 'rb') as f:
                return pickle.load(f)

res = load_obj('out')
lists = sorted(res.items())
x, y = zip(*lists)
plt.plot(x, y)
plt.show()
```