IMA208
Range scans to meshes

Alice Valença De Lorenci

April 5, 2023

The objective of this practical work was to reconstruct the surface of an object from a point cloud. We worked with the *Bunny.xyz* and *Bimba.xyz* point clouds.

With that objective, the following code was implemented in *Python*:

```
import numpy as np
from scipy.spatial import Delaunay
from scipy.spatial import distance

# Specify object
filename = "Bimba"

# Set threshold
if filename == "Bunny":
    t = 0.002
elif filename == "Bimba":
    t = 0.017
else:
    print("Manually select a threshold (default t=0.1)")
    t = 0.1

# (3D)
# Open the file and read the vertices as strings
with open("{}.xyz".format(filename), "r") as f:
    vertex_strings = f.readlines()

# Convert the vertex strings to a NumPy array of shape (N, 3)
points3D = np.zeros((len(vertex_strings), 3))
for i, vertex_str in enumerate(vertex_strings):
    vertex_arr = [float(coord) for coord in vertex_str.strip().split()]
    points3D[i] = vertex_arr


# Obtain Delaunay triangulation
tri = Delaunay(points3D)
kept_tri = []

ntri = 4*len(tri.simplices) # used to keep track of progress
count = 0

# Delaunay filtering
for tetra in tri.simplices:
    for k in range(len(tetra)):
        # vertices of the triangle
        v1, v2, v3 = points3D[tetra[k%4]],points3D[tetra[(k+1)%4]],points3D[tetra[(
    k+2)%4]]
```

```
        # length of edges
        a = np.linalg.norm( v1-v2 )
        b = np.linalg.norm( v1-v3 )
        c = np.linalg.norm( v2-v3 )
        # radius of the circumcircle
        r = (a*b*c) / np.sqrt( (a+b+c)*(b+c-a)*(c+a-b)*(a+b-c) )
        # filtering
        if( r<t ):
            kept_tri.append( [v1, v2, v3] )
        # keep track of progress
        count = count+1
        if( count%1000 == 0 ):
            print("\r{}/{}".format(count, ntri), end="")

print('\r', end="")

# Save selected triangles to output .stl file
with open("{}_t{}.stl".format(filename, t), "w") as f:
    f.write("solid output")
    for triangle in kept_tri:
        f.write("facet normal 0 0 0\n")
        f.write("\touter loop\n")
        for vertex in triangle:
            f.write("\t\tvertex {} {} {}\n".format(vertex[0], vertex[1], vertex[2])
    )
        f.write("\tend loop\n")
        f.write("end facet\n")
    f.write("endsolid output")
```

First of all, we obtain the Delaunay triangulation of the point cloud. It is known that it contains the approximated surface of the object we are trying to reconstruct, therefore it is necessary to select only the triangles that correspond to that surface.

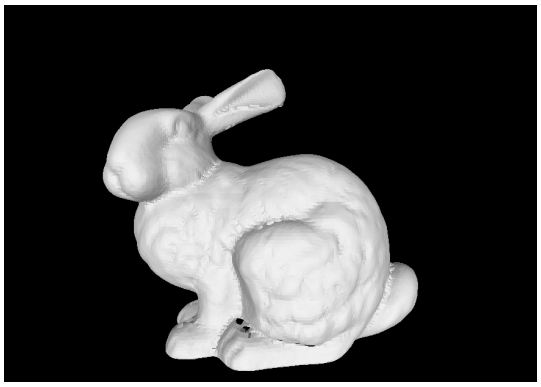
A very simple algorithm that does that consists in setting a threshold t and computing the radius R of the circumcircle for each triangle, if $R < t$, we keep the triangle, else, it is discarded. It is necessary to adjust the value of the threshold for each point cloud we work with.

For the *Bunny.xyz* point cloud (Figure 1a), the best value for the threshold was $t = 0.002$ (Figure 1c).

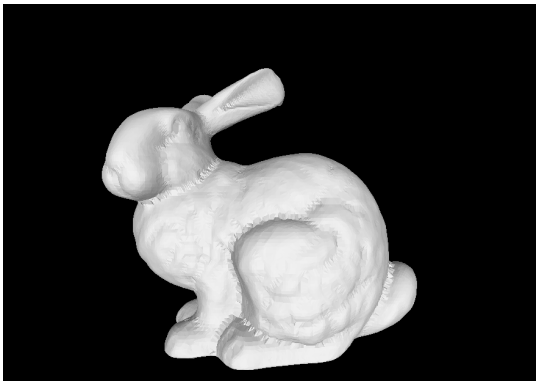
For the *Bimba.xyz* point cloud (Figure 2a), it wasn't possible to obtain an acceptable reconstruction using a static threshold, either the reconstruction was too coarse (Figure 2d), specially around the neck, or it had holes in it (Figure 2b). The best trade-off was obtained for $t = 0.015$ (Figure 2c).

In this case, an alternative would be to use an adaptive threshold. For instance, the threshold could depend on the density of the point cloud in the neighborhood of the triangle: a lower threshold for denser regions of the point cloud and a higher threshold for sparser regions.

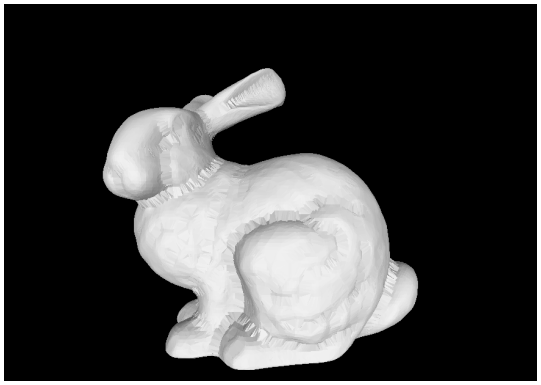
(a) Point cloud



(b) $t = 0.0015$

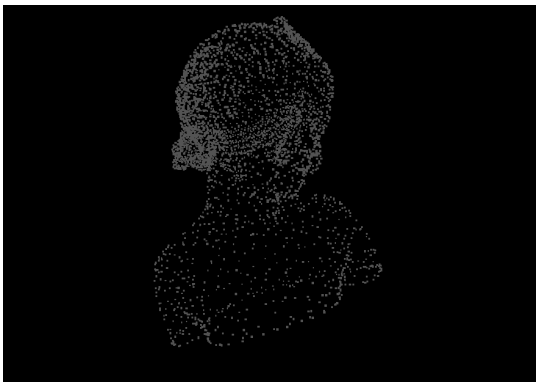


(c) $t = 0.002$

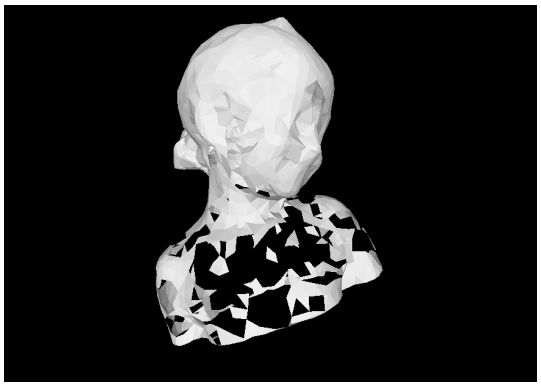


(d) $t = 0.003$

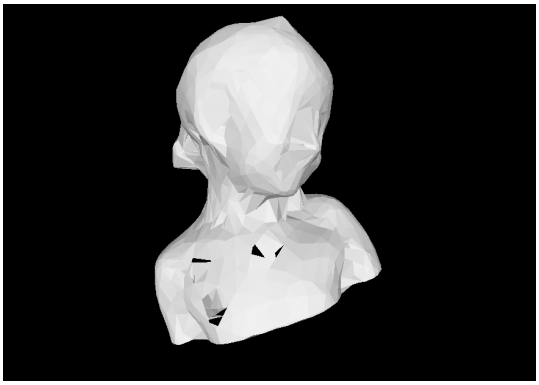
Figure 1: Bunny



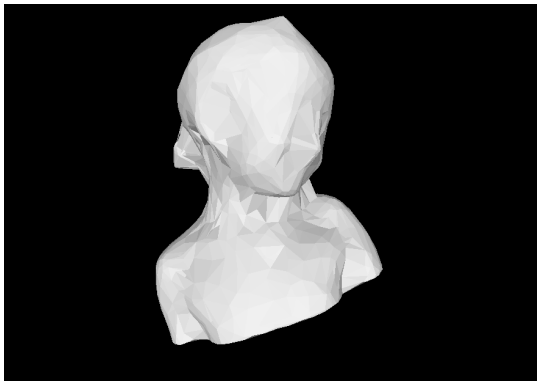
(a) Point cloud



(b) $t = 0.01$



(c) $t = 0.015$



(d) $t = 0.017$

Figure 2: Bimba